UNIVERSIDAD DE CÓRDOBA

Departamento de Informática y Análisis Numérico

Programa de doctorado en computación avanzada, energía y plasmas

Modelos metaheurísticos para el soporte a la decisión en el proceso de construcción de software

Metaheuristic models for decision support in the software construction process

Memoria de Tesis presentada por

Aurora Ramírez Quesada

como requisito para optar al grado

de Doctor en Informática

Directores

Dr. José Raúl Romero Salguero

Dr. Sebastián Ventura Soto

Córdoba, septiembre de 2018

UNIVERSITY OF CÓRDOBA

Department of Computer Science and Numerical Analysis

Metaheuristic models for decision support in the software construction process

A Thesis submitted by

Aurora Ramírez Quesada

in fulfilment of the requirements for
the degree of Doctor in Computer Science

Supervisors

Dr. José Raúl Romero Salguero

Dr. Sebastián Ventura Soto

Córdoba, September 2018

La memoria titulada *"Modelos metaheurísticos para el soporte a la decisión en el proceso de contrucción de software"*, que presenta Aurora Ramírez Quesada para optar al grado de Doctor en el marco del programa de doctorado "Computación avanzada, energía y plasmas", recopila un trabajo original de investigación realizado en el Departamento de Informática y Análisis Numérico de la Escuela Politécnica Superior de la Universidad de Córdoba. Dicho trabajo ha sido realizado bajo la dirección de Dr. José Raúl Romero Salguero y Dr. Sebastián Ventura Soto cumpliendo, a su juicio, los requisitos exigidos a este tipo de trabajos y respetando los derechos de otros autores a ser citados, cuando se han utilizado sus resultados o publicaciones.

Córdoba, septiembre de 2018

La candidata:

Fdo.: Aurora Ramírez Quesada

Los directores:

Fdo.: Dr. José Raúl Romero Salguero

Fdo.: Dr. Sebastián Ventura Soto

Tesis con mención internacional

Esta tesis cumple los criterios establecidos por la Universidad de Córdoba para la obtención del Título de Doctor con Mención Internacional:

1. Estancia predoctoral mínima de 3 meses fuera de España en una institución de enseñanza superior o centro de investigación de prestigio, cursando estudios o realizando trabajos de investigación relacionados con la tesis doctoral:

   Department of Computer Science and Creative Technologies, Faculty of Environment and Technology, University of the West of England, Bristol, United Kingdom. Responsable de la estancia: **Dr. Christopher L. Simons**, Senior Lecturer.

2. La tesis cuenta con el informe previo de dos doctores o doctoras expertos y con experiencia investigadora acreditada pertenecientes a alguna institución de educación superior o instituto de investigación distinto de España:

   a. Dr. Virgilijus Sakalauskas, Professor. Dept. Informatics, Kaunas Faculty, Vilnius University, Lithuania.

   b. Dr. Francisco Servant, Assistant Professor. Dept. Computer Science, Virginia Tech., Blacksburg, Virginia, United States of America.

3. Entre los miembros del tribunal evaluador de la tesis se encuentra un doctor procedente de una institución de educación superior distinto de España y diferente del responsable de la estancia predoctoral:

   Dr. Robert Feldt, Professor. Dept. Computer Science and Engineering, Chalmers University of Technology, Gothenburg, Sweden.

4. Parte de la tesis doctoral se ha redactado y presentado en dos idiomas, castellano e inglés.

Córdoba, septiembre de 2018

La candidata:

Fdo.: Aurora Ramírez Quesada

Tesis Doctoral subvencionada por el programa de Formación del Profesorado Universitario (FPU) del Ministerio de Educación, Cultura y Deportes (referencia **FPU13/01466**), convocatoria publicada en el B.O.E. №279 de 21 de noviembre de 2013 y resuelta en el B.O.E. №215 de 4 de septiembre de 2014.

La estancia para la obtención de la mención internacional ha sido financiada por el mencionado programa (referencia **EST16/00143**), conforme a la convocatoria publicada en el B.O.E. №14 de 17 de enero de 2017 y resuelta el 4 de abril de 2017.

Asimismo, esta Tesis Doctoral ha sido parcialmente subvencionada por el Ministerio de Ciencia y Tecnología mediante el proyecto **TIN2011-22408**, el Ministerio de Economía y Competitivad (proyectos **TIN2014-55252-P** y **TIN2017-83445-P**), la Red de Excelencia en Ingeniería de Software basada en búsqueda (**TIN2015-71841-REDT**), y fondos FEDER.

Agradecimientos

"No duty is more urgent than that of returning thanks."
James Allen

Llegando al final de una larga etapa de formación, si es que eso termina en algún momento, me doy cuenta de lo afortunada que he sido por poder compartirla con tantas personas. Una página no es suficiente para expresar mi agradecimiento a todas ellas, pero lo intentaré.

A la casualidad, que cruzó a Raúl en mi camino hace ya más de 10 años, por haberme permitido descubrir a un atento tutor, un excelente profesor, un magnífico director y, sobre todo, una gran persona y mejor amigo. Raúl, te doy las gracias por tu entrega, reflejada en cada minuto de cada reunión frente a la desafiante pizarra en blanco, cada palabra en los incontables correos electrónicos, o cada coma en rojo en cada manuscrito que con tanto esmero revisas, porque todo ello me ha convertido en la investigadora que soy hoy. A Sebastián, por abrirle las puertas del laboratorio a aquella tímida estudiante, y darle la oportunidad de desarrollar su trabajo en él. Tu experiencia y apoyo constante también han sido claves en su consecución.

A mis compañeros de laboratorio, con quienes he tenido la suerte de compartir el día a día. A esa *primera generación* (Juan Ignacio, Juan Luis, José María, Alberto y Oscar) porque, cada uno a vuestra manera, me habéis servido de ejemplo. A la *segunda generación* (José María, Rubén y Rafa), por su agradable compañía en los últimos años. Os deseo que disfrutéis lo que os queda de camino. Al resto de miembros del grupo KDIS por su apoyo, especialmente a Carlos, por tener siempre un momento para explicarme los entresijos del misterioso mundo de las metaheurísticas.

También quiero mostrar mi agradecimiento al departamento de Informática y Análisis Numérico y a buena parte de la Escuela Politécnica Superior, cuyos profesores han mostrado interés en mi trabajo cada vez que nos cruzábamos por un pasillo. No me puedo olvidar de la magnífica comunidad SBSE en España, cuyas ideas han sido fuente de inspiración a lo largo de estos años. A Chris Simons, *the perfect British host*, por su amabilidad y ayuda durante mi estancia en Bristol, y con quien espero poder seguir debatiendo ideas brillantes.

Quiero también agradecer a todos los compañeros y amigos con los que he ido compartiendo mis distintas etapas universitarias. Vosotros me recordáis que existe informática más allá de la investigación, e investigación más allá de la informática. A mis amigos de siempre, porque me recuerdan que existe vida fuera de ambas.

Finalmente, a mi pequeña pero gran familia, por su infinito cariño y apoyo. A mis padres, Mateo y Pilar, por haberme bindrado la mejor educación posible y estar siempre a mi lado de forma incondicional. A mi hermana, Estrella, con quien más y mejores momentos he compartido desde que tengo uso de razón, porque siempre estás ahí para evadirnos juntas del doctorado. Estoy segura de que muy pronto terminarás el tuyo con todos los honores que mereces.

Gracias de todo corazón.
Aurora

Resumen

En la actualidad, los ingenieros software no solo tienen la responsabilidad de construir sistemas que desempeñen una determinada funcionalidad, sino que cada vez es más importante que dichos sistemas también cumplan con requisitos no funcionales como alta disponibilidad, eficiencia o seguridad, entre otros. Para lograrlo, los ingenieros se enfrentan a un proceso continuo de decisión, pues deben estudiar las necesidades del sistema a desarrollar y las alternativas tecnológicas existentes para implementarlo. Todo este proceso debe estar encaminado a la obtención de sistemas software de gran calidad, reutilizables y que faciliten su mantenimiento y modificación en un escenario tan exigente y competitivo.

La ingeniería del software, como método sistemático para la construcción de software, ha aportado una serie de pautas y tareas que, realizadas de forma disciplinada y adaptadas al contexto de desarrollo, posibilitan la obtención de software de calidad. En concreto, el proceso de análisis y diseño del software ha adquirido una gran importancia, pues en ella se concibe la estructura del sistema, en términos de sus bloques funcionales y las interacciones entre ellos. Es en este momento cuando se toman las decisiones acerca de la arquitectura, incluyendo los componentes que la conforman, que mejor se adapta a los requisitos, tanto funcionales como no funcionales, que presenta el sistema y que claramente repercuten en su posterior desarrollo. Por tanto, es necesario que el ingeniero analice rigurosamente las alternativas existentes, sus implicaciones en los criterios de calidad impuestos y la necesidad de establecer compromisos entre ellos. En este contexto, los ingenieros se guían principalmente por sus habilidades y experiencia, por lo que dotarles de métodos de apoyo a la decisión representaría un avance significativo en el área.

La aplicación de técnicas de inteligencia artificial en este ámbito ha despertado un gran interés en los últimos años. En particular, la inteligencia artificial ha encontrado en la ingeniería del software un ámbito de aplicación complejo, donde diferentes técnicas pueden ayudar a conseguir la semi-automatización de tareas tradicionalmente realizadas de forma manual. De la unión de ambas áreas surge la denominada ingeniería del software basada en búsqueda, que propone la reformulación de las actividades propias de la ingeniería del software como problemas de optimización. A continuación, estos problemas podrán ser resueltos mediante técnicas de búsqueda como las metaheurísticas. Este tipo de técnicas se caracterizan por explorar el espacio de posibles soluciones de una manera "inteligente", a menudo simulando procesos naturales como es el caso de los algoritmos evolutivos.

A pesar de ser un campo de investigación muy reciente, es posible encontrar propuestas para automatizar una gran variedad de tareas dentro del ciclo de vida del

software, como son la priorización de requisitos, la planificación de recursos, la refactorización del código fuente o la generación de casos de prueba. En el ámbito del análisis y diseño de software, cuyas tareas requieren de creatividad y experiencia, conseguir una automatización completa resulta poco realista. Es por ello por lo que la resolución de sus tareas mediante enfoques de búsqueda debe ser tratada desde la perspectiva del ingeniero, promoviendo incluso la interacción con ellos. Además, el alto grado de abstracción de algunas de sus tareas y la dificultad de evaluar cuantitavimente la calidad de un diseño software, suponen grandes retos en la aplicación de técnicas de búsqueda durante las fases tempranas del proceso de construcción de software.

Esta tesis doctoral busca realizar aportaciones significativas al campo de la ingeniería del software basada en búsqueda y, más concretamente, al área de la optimización de arquitecturas software. Aunque se están realizando importantes avances en este área, la mayoría de propuestas se centran en la obtención de arquitecturas de bajo nivel o en la selección y despliegue de artefactos software ya desarrollados. Por tanto, no existen propuestas que aborden el modelado arquitectónico a un nivel de abstracción elevado, donde aún no existe un conocimiento profundo sobre cómo será el sistema y, por tanto, es más difícil asistir al ingeniero. Como problema de estudio, se ha abordado principalmente la tarea del *descubrimiento de arquitecturas software basadas en componentes*. El objetivo de este problema consiste en abstraer los bloques arquitectónicos que mejor definen la estructura actual del software, así como sus interacciones, con el fin de facilitar al ingeniero su posterior análisis y mejora.

Durante el desarrollo de esta tesis doctoral se ha explorado el uso de una gran variedad de técnicas de búsqueda, estudiando su idoneidad y realizando las adaptaciones necesarias para hacer frente a los retos mencionados anteriormente. La primera propuesta se ha centrado en la formulación del descubrimiento de arquitecturas como problema de optimización, abordando la representación computacional de los artefactos software que deben ser modelados y definiendo medidas software para evaluar su calidad durante el proceso de búsqueda. Además, se ha desarrollado un primer modelo basado en algoritmos evolutivos mono-objetivo para su resolución, el cual ha sido validado experimentalmente con sistemas software reales. Dicho modelo se caracteriza por ser comprensible y flexible, pues sus componentes han sido diseñados considerando estándares y herramientas del ámbito de la ingeniería del software, siendo además configurable en función de las necesidades del ingeniero.

A continuación, el descubrimiento de arquitecturas ha sido tratado desde una perspectiva multiobjetivo, donde varias medidas software, a menudo en conflicto, deben ser simultáneamente optimizadas. En este caso, la resolución del problema se ha llevado a cabo mediante ocho algoritmos del estado del arte, incluyendo propuestas

recientes del ámbito de la optimización de muchos objetivos. Tras ser adaptados al problema, estos algoritmos han sido comparados mediante un extenso estudio experimental con el objetivo de analizar la influencia que tiene el número y la elección de las métricas a la hora de guiar el proceso de búsqueda. Además de realizar una validación del rendimiento de estos algoritmos siguiendo las prácticas habituales del área, este estudio aporta un análisis detallado de las implicaciones que supone la optimización de múltiples objetivos en la obtención de modelos de soporte a la decisión.

La última propuesta en el contexto del descubrimiento de arquitecturas software se centra en la incorporación de la opinión del ingeniero al proceso de búsqueda. Para ello se ha diseñado un mecanismo de interacción que permite al ingeniero indicar tanto las características deseables en las soluciones arquitectónicas (preferencias positivas) como aquellos aspectos que deben evitarse (preferencias negativas). Esta información es combinada con las medidas software utilizadas hasta el momento, permitiendo al algoritmo evolutivo adaptar la búsqueda conforme el ingeniero interactúe. Dadas las características del modelo, su validación se ha realizado con la participación de ingenieros con distinta experiencia en desarrollo software, a fin de demostrar la idoneidad y utilidad de la propuesta.

En el transcurso de la tesis doctoral, los conocimientos adquiridos y las técnicas desarrolladas también han sido extrapolados a otros ámbitos de la ingeniería del software basada en búsqueda mediante colaboraciones con investigadores del área. Cabe destacar especialmente la formalización de una nueva disciplina transversal, denominada ingeniería del software basada en búsqueda interactiva, cuyo fin es promover la participación activa del ingeniero durante el proceso de búsqueda. Además, se ha explorado la aplicación de algoritmos de muchos objetivos a un problema clásico de la computación orientada a servicios, como es la composición de servicios web.

Abstract

Nowadays, software engineers have not only the responsibility of building systems that provide a particular functionality, but they also have to guarantee that these systems fulfil demanding non-functional requirements like high availability, efficiency or security. To achieve this, software engineers face a continuous decision process, as they have to evaluate system needs and existing technological alternatives to implement it. All this process should be oriented towards obtaining high-quality and reusable systems, also making future modifications and maintenance easier in such a competitive scenario.

Software engineering, as a systematic method to build software, has provided a number of guidelines and tasks that, when done in a disciplinarily manner and properly adapted to the development context, allow the creation of high-quality software. More specifically, software analysis and design has acquired great relevance, being the phase in which the software structure is conceived in terms of its functional blocks and their interactions. In this phase, engineers have to make decisions about the most suitable architecture, including its constituent components. Such decisions are made according to the system requirements, either functional or non-functional, and will have a great impact on its future development. Therefore, the engineer has to rigorously analyse existing alternatives, their implications on the imposed quality criteria and the need of establishing trade-offs among them. In this context, engineers are mostly guided by their own capabilities and experience, so providing them with decision support methods would represent a significant contribution.

The application of artificial intelligent techniques in this area has experienced a growing interest in the last years. Particularly, software engineering represents a complex application domain to artificial intelligence, whose diverse techniques can help in the semi-automation of tasks traditionally performed manually. The union of both fields has led to the appearance of search-based software engineering, which proposes reformulating software engineering activities as optimisation problems. For their resolution, search techniques like metaheuristics can be then applied. This type of technique performs an "intelligent" exploration of the space of candidate solutions, often inspired by natural processes as happens with evolutionary algorithms.

Despite the novelty of this research field, there are proposals to automate a great variety of tasks within the software lifecycle, such as requirement prioritisation, resource planning, code refactoring or test case generation. Focusing on analysis and design, whose tasks require creativity and experience, trying to achieve full automation is not realistic. Therefore, solving design tasks by means of search approaches

should be oriented towards the engineer's perspective, even promoting their interaction. Furthermore, design tasks are also characterised by a high level of abstraction and the difficulty of quantitatively evaluating design quality. All these aspects represent key challenges for the application of search techniques in early phases of the software construction process.

The aim of this Ph.D. Thesis is to make significant contributions in search-based software engineering and, specially, in the area of software architecture optimisation. Although it is an area in which significant progress is being done, most of the current proposals are focused on generating low-level architectures or selecting and deploying already developed artefacts. Therefore, there is a lack of proposals dealing with architectural modelling at a high level of abstraction. At this level, engineers do not have a deep understanding of the system yet, meaning that assisting them is even more difficult. As case study, the *discovery of component-based software architectures* has been primary addressed. The objective for this problem consists in the abstraction of the architectural blocks, and their interactions, that best define the current structure of a software system. This can be viewed as the first step an engineer would perform in order to further analyse and improve the system architecture.

In this Ph.D. Thesis, the use of a great variety of search techniques has been explored. The suitability of these techniques has been studied, also making the necessary adaptations to cope with the aforementioned challenges. A first proposal has been focused on the formulation of software architecture discovery as an optimisation problem, which consists in the computational representation of its software artefacts and the definition of software metrics to evaluate their quality during the search process. Moreover, a single-objective evolutionary algorithm has been designed for its resolution, which has been validated using real software systems. The resulting model is comprehensible and flexible, since its components have been designed under software engineering standards and tools and are also configurable according to engineer's needs.

Next, the discovery of software architectures has been tackled from a multi-objective perspective, in which several software metrics, often in conflict, have to be simultaneously optimised. In this case, the problem is solved by applying eight state-of-the-art algorithms, including some recent many-objective approaches. These algorithms have been adapted to the problem and compared in an extensive experimental study, whose purpose is to analyse the influence of the number and combination of metrics when guiding the search process. Apart from the performance validation following

usual practices within the field, this study provides a detailed analysis of the practical implications behind the optimisation of multiple objectives in the context of decision support.

The last proposal is focused on interactively including the engineer's opinion in the search-based architecture discovery process. To do this, an interaction mechanism has been designed, which allows the engineer to express desired characteristics for the solutions (positive preferences), as well as those aspects that should be avoided (negative preferences). The gathered information is combined with the software metrics used until the moment, thus making possible to adapt the search as the engineer interacts. Due to the characteristics of the proposed model, engineers of different expertise in software development have participated in its validation with the aim of showing the suitability and utility of the approach.

The knowledge acquired along the development of the Thesis, as well as the proposed approaches, have also been transferred to other search-based software engineering areas as a result of research collaborations. In this sense, it is worth noting the formalisation of interactive search-based software engineering as a cross-cutting discipline, which aims at promoting the active participation of the engineer during the search process. Furthermore, the use of many-objective algorithms has been explored in the context of service-oriented computing to address the so-called web service composition problem.

Preface

The Spanish legislation for Ph.D. studies, RD 99/2011, published the 28^{th} of January of 2011 (BOE-A-2011-2541), grants each Spanish University competencies to establish the necessary supervision and evaluation procedures to guarantee the quality of Ph.D. Theses. As unique requirement for the defence, this national regulation indicates that the manuscript should be accompanied by a document detailing the complementary learning activities carried out by the student.

Accordingly, the University of Córdoba has a specific regulation for Ph.D. studies, approved by its governing board the 21^{th} of December of 2011. This regulation establishes two different modalities to elaborate the manuscript that the student, under the supervision of one or more Ph.D. advisors, has to present at the end of his/her doctorate studies. This Ph.D. Thesis follows the modality described in the article no. 24 of the aforementioned regulation, referred as *Ph.D. Thesis as a compendium of publications*. According to that article, the Ph.D. Thesis can be presented as a compendium of, at least, three research articles published (or accepted for publication) in research journals of high quality, i.e. appearing in the first three quartiles of the Journal Citation Reports (JCR). If such a requirement is fulfilled, the manuscript has to include: an introduction to justify the thematic cohesion of the Ph.D. Thesis; the hypotheses and objectives to be achieved, and how they are associated to the publications; full copy of the publications, and conclusions.

Following these guidelines, this Ph.D. Thesis is organised as described next. Firstly, an introductory part is divided into five chapters. More specifically, Chapter 1 presents the background and state of the art of the research areas in which this Ph.D. Thesis is framed. Next, the motivation, objectives and hypotheses are detailed in Chapter 2. Chapter 3 explains the research methodology, while an overview of the obtained results is presented in Chapter 4. Lastly, Chapter 5 discusses conclusions and future work. The second part of the document is comprised of three chapters. Chapter 6 includes the three main publications derived from this Ph.D. Thesis. Chapter 7 compiles other journal publications associated to this Ph.D. Thesis. Finally, Chapter 8 provides the list of conference publications.

Contents

List of Figures

List of Tables

List of Acronyms

ADL architecture description language

AI artificial intelligence

ACO ant colony optimisation

CBSE component-based software engineering

COTS commercial-off-the-shelf

DSS decision support system

EA evolutionary algorithm

EC evolutionary computation

HC hill climbing

IEC interactive evolutionary computation

LS local search

MA memetic algorithm

MaOEA many-objective evolutionary algorithm

MaOO many-objective optimisation

MaOP many-objective problem

MCDM multiple criteria decision making

MDE model-driven engineering

MOEA multi-objective evolutionary algorithm

MOO multi-objective optimisation

MOP multi-objective problem

MVC model-view-controller

PF Pareto front

PS Pareto set

PSO particle swarm optimisation

QoS quality-of-service

QoSWSC QoS-aware web service composition

RQ research question

SA simulated annealing

SE software engineering

SI swarm intelligence

SBSE search-based software engineering

SBSD search-based software design

SLR systematic literature review

SPL software product line

SOA service-oriented architecture

TS tabu search

UML unified modelling language

Part I

Introduction

1

Background

"Learning never exhausts the mind".
Leonardo da Vinci

This chapter presents the fundamentals and state of the art of the research areas in which this Ph.D. Thesis is founded. More precisely, the conceptual framework underlying the design of software architectures is firstly described. Decision support methods for the architecting process are also covered. Then, an introduction to search techniques is presented, including an overview of metaheuristics with special focus on evolutionary computation (EC). Two advanced approaches, i.e. multi-objective optimisation (MOO) and interactive optimisation, are detailed next. Lastly, search-based software engineering (SBSE) is explored in depth in order to analyse the current state of the field regarding the application of search techniques to address software design problems. A historical perspective of SBSE methods for software architecture optimisation is also provided.

1.1. Software architectures

1.1.1. Foundations and definitions

The specification of abstract descriptions of software systems has always been a central part of the software development process [73]. However, the lack of specific methodologies has clearly hampered design traceability and knowledge transfer at the beginnings of software engineering (SE). The increasing complexity of software systems has led to the appearance of a more disciplined approach, allowing to establish a common terminology to reason about their high-level structure. Software architecting is now a well-established practice within the software industry, with specialised engineers, description languages and modelling tools. According to the *ISO/IEC/IEEE Std. 42010:2011 System and software engineering – Architecture description* [94], a software architecture is defined as follows:

> A **software architecture** represents "the fundamental concepts or properties of a system in its environment embodied in its elements, relationships, and in the principles of its design and evolution".

Therefore, the architectural analysis of a software system not only represents an essential activity during early software conception, but also guides its subsequent development. The aforementioned definition also refers to the external environment in which the system will operate, which is equally important than its internal structure [57]. Hence, software architectures can be viewed at two different levels: *macro-architecture*, which concerns the system environment, and a *micro-architecture*, which dictates how the system is internally organised [57].

Focusing on this latter perspective, software architectures act as a bridge between requirements and implementation [73]. In this sense, a software architecture provides a high-level description of the system that allows engineers to specify how requirements are to be satisfied and what properties the system has to exhibit. According to Garlan [73], software architectures are fundamental to the following aspects of software development:

1. *Understanding*, since they describe the system at a level of abstraction that makes software comprehension and reasoning easier;

2. *Reuse*, due to the fact that architectural solutions result in independent components, which are often based on recurrent patterns;

3. *Construction*, as the architecture specifies the principal functional blocks and existing dependencies between them;

4. *Evolution*, for which the architecture separates the functionalities from the mechanisms to manage their interactions, which are subject to change in the future;

5. *Analysis*, architectures being an important input for the assessment of non-functional properties, conformance to styles and constraint satisfaction; and,

6. *Management*, since designing a good architecture can lead to cost and effort savings, which are crucial for the success of complex industrial systems.

Languages and architectural styles

Formal notations to model and manage software architectures are frequently adopted in academia and industry. In this sense, the *ISO Std. 42010* [94] provides the following definition:

> An **architecture description language (ADL)** "is any form of expression for use in architecture descriptions".

More specifically, an ADL specifies a conceptual framework and a concrete syntax to specify the architecture [73]. ADLs must be simple, interpretable, understandable and not necessarily graphic, though they are often supported by tools to create, visualise and analyse architectural models [94]. An example of a generic ADL is Acme, whist other ADLs are specific, such as AADL (Architecture Analysis and Design Language) and EADL (Embedded Architecture Description Language) for embedded systems or Darwin for distributed systems. More general, the unified modelling language (UML) [138] supports now modelling architectural concepts beyond deployment aspects, the only view considered in its first version.

Due to the variety of purposes for which software systems are now conceived and the broad range of possible technologies to bring them into life, architectural solutions

often conform to a particular *style* and adopt *patterns* to successfully achieve system goals. Both concepts are highly relevant in today's industrial practice. On the one hand, architectural styles, such as data flow architectures or data-centred architectures, provide a common vocabulary to interpret an architecture [2]. On the other hand, architectural patterns like model-view-controller (MVC) are reusable solutions for recurrent problems that appear within a particular context [32]. As opposed to them, *architectural bad smells* are anti-patterns that appear as consequence of design decisions that negatively impact quality properties, such as understandability and maintainability [72].

Component-based software architectures

Component-based software architectures represent a particular type of software architecture whose distinctive characteristic is the reuse of functionality. A well-established definition of software component is provided by C. Szyperski [185]:

> "A **software component** is a unit of composition with contractually specified interfaces and explicit context dependencies only. A software component can be deployed independently and is subject to third-party composition". Each **interface** is "a set of named operations that can be invoked by clients".

Components are linked by means of **connectors**, which serve to specify that one part of the system is providing the services that other parts need to operate [138]. In component-based software engineering (CBSE), the architect is responsible for identifying components, assigning responsibilities to them and establishing how they will collaborate through well-defined interfaces [76, 185]. This way, the system is constructed by assembling independent and reusable components that all together will provide the required functionality. As components are abstract units, they can represent a variety of artefacts, including modules or packages in object-oriented systems, services in cloud environments or distributed objects in distributed systems. This characteristic also implies that they remain independent of the specific languages and technologies that will be later selected for development and deployment. In fact, a complementary decision for the architect to make is whether a component should be built in-house or it can be acquired from specialised repositories. In this

sense, *commercial-off-the-shelf (COTS) components* offer already implemented and tested functionality that can greatly reduce development effort but at the expense of integration costs. Also, architects should be aware of *legacy software*, i.e. old parts of the system that are critical to the business and should remain operative [153]. These legacy systems are non-replaceable due to high rebuilding cost, inextensible design, lack of proper documentation or obsolete hardware, so the proposed architectural solution should provide effective ways for integration and communication.

1.1.2. The architecting process

The relevance and cross-cutting nature of software architectures imply that software architects play a pivotal role within the software project. They have to interact with stakeholders, lead the design team and be in permanent communication with project managers to guarantee project success [76]. Furthermore, their expertise, experience and *know-how* become indispensable when it comes to deal with complex systems for which innovative solutions are required. A software architect is involved in an iterative process comprised of three main steps [76]:

1. *Architecture requirements definition*, which is aimed at producing a specification of those requirements that are relevant to the software architecture.
2. *Architecture design*, which includes the identification of components, and the allocation of responsibilities.
3. *Architecture validation*, which assesses that the proposed architecture fits its purpose according to project requirements and constraints.

Architecture identification and recovery

Within the architecture design phase, component modelling can be carried out with a variety of methodologies. Two relevant activities for which supporting methods have been proposed are *component identification* and *architecture recovery*. On the one hand, component identification methods are applied to derive a partition of the system functionalities from the requirements specification at the beginning of the development process [23]. Existing methods are often top-down approaches but they can differ in how components are defined and which are the particular goals to be

pursued. Furthermore, a distinction has to be made regarding how these methods are described, i.e. from general recommendations to more formal methodologies [23]. Other approaches rely on metrics like coupling and cohesion to guide the creation of component-based designs [109].

On the other hand, architecture recovery concerns the reconstruction of the system architecture from low-level artefacts, specially code [59]. This re-engineering process can be required during maintenance tasks, since architects need to comprehend the actual structure of the system in order to extend or adapt it to new requirements or contexts. Due to uncontrolled changes, design documentation often become incoherent, meaning that code represents the main source of information for architects. Systematic manual inspection [188], semi-automated methods based on clustering [121] and combined approaches [142] can help engineers to retrieve some high-level design entities. This is still an active focus of research and current proposals are studying what code elements can be mapped onto architectural elements and how they influence recovery techniques [43, 115, 183].

Architecture evaluation

During validation, architects verify the fulfilment of quality properties and identify potential risks [57]. Even though precise quality estimations are not possible at such an early stage of the project, software architecture analysis methods can provide evidence of the effects of the architecture on quality. Evaluation methods at the architectural level are classified in two main categories, namely *questioning techniques* and *measuring techniques*, which actually can complement each other [18]. The former are mostly based on qualitative assessment by means of scenarios and check lists, whereas the latter require software metrics and simulations to quantify the quality of the architecture. Nevertheless, current analysis methods are mostly manual processes that strongly rely on the experience of the architect.

Focusing on measuring techniques, the definition of software metrics able to accurately reflect quality properties is still a paramount concern within the SE community. Progress has been made with the definition of the SQuaRE quality model for system and software products in the *ISO/IEC Std. 25010– Systems and software quality models* [93], but it does not provide specific metrics to measure quality properties. Similarly, the upcoming standard for architecture evaluation (*ISO/IEC Std.*

42030 – Architecture evaluation), whose publication is expected during 2018, seems to be focused on desired characteristics of evaluation methods and how they can determine the extent to which stakeholders' concerns are addressed [126].

Given that component-based software architectures rely on composition principles in order to allow the reuse of functionalities, ease of maintenance becomes the primary quality attribute. In this sense, the *ISO/IEC Std. 25010* provides a formal definition of *maintainability*:

> Software **maintainability** refers to "the degree to which the software product can be modified. Modifications may include corrections, improvements or adaptation of the software to changes in environment, and in requirements and functional specifications".

Among the characteristics in which software maintainability is decomposed, the following three characteristics are essential in CBSE:

- *Modularity*, which is defined as "the degree to which a system is composed of discrete components such that a change to one component has minimal impact on other components".
- *Reusability*, which establishes "the degree to which an asset can be used in more than one software system or in building other assets".
- *Analysability*, which determines "the degree to which the parts of the software to be modified can be identified".

In order to measure these and other qualitative aspects of software architectures, different metrics have been extensively investigated in the literature. First attempts consist in the specialisation of general quality models for software architectures [113], the adaptation of object-oriented metrics [194] or the proposal of novel metric suites [132]. Other authors focus their studies on the assessment of certain properties, such as modularity [167] or adaptability [148]. Within CBSE, it is also possible to find compilations of metrics for components [1, 117, 132], as well as specific metrics and evaluation methods for relevant properties, such as reusability [201], analysability [29] or usability of COTS components [22].

1.1.3. Decision support for architecture design

Decision making is an intrinsic characteristic of the architecting process, since software architects have to conceive different solutions and choose the best alternative according to functional requirements, non-functional requirements and business goals [6, 64]. Therefore, architecture design can be studied from the perspective of multiple criteria decision making (MCDM). In this sense, the International MCDM Society [92] provides the following definition [89]:

> **MCDM** encompasses "the study of methods and procedures by which the concerns about multiple, usually conflicting, criteria can be formally incorporated into the management planning process".

In this context, a decision-making scenario is characterised by the set of *alternatives*, each one representing a different choice to the *decision maker*, and a number of *decision criteria*, that establish the different views from which alternatives can be evaluated [190]. Taking both elements as input, a *decision-making technique* provides a systematic process to choose the best alternative among the existing ones with respect to decision criteria [64, 190]. A decision support system (DSS) can be then constructed to provide the MCDM methods as a computer-based information system [131]. When these decision criteria can be numerically quantified, MCDM techniques perform the following steps [190]:

1. Formal definition of relevant criteria and alternatives.
2. Determination of the relative importance of the decision criteria.
3. Numerical evaluation of alternatives with respect to the decision criteria.
4. Prioritisation of the alternatives on the basis of the numerical assessment.

Several authors have pointed out the benefits that a systematic decision-making process could bring to the architecting process [64, 134]. Such a process would help architects to manage trade-offs among conflicting goals, deal with the inherent uncertainty of early analysis, and evaluate potential consequences of their decisions.

Also, this process could serve to capture the rationale behind architectural decisions, making them explicit and well-documented. In fact, the lack of support to architectural knowledge management has been identified as a possible aspect limiting broader adoption of decision-making methods by the software industry [49]. Furthermore, MCDM methods could involve multiple stakeholders or design team members within the process to get closer to real design scenarios [162], for which *group decision-making* techniques could be considered.

Examples of MCDM methods to support different activities related to architecture design can be found in the literature. Svahnberg et al. have proposed a method to identify the best architecture among a set of preliminary solutions [184]. The analytic hierarchy process (AHP) is applied to prioritise these solutions according to quality attributes and the opinions of multiple stakeholders. AHP analyses all possible pairwise comparisons to rate the alternatives, and is a popular technique within other areas of SE, such as requirement prioritisation [21]. Another DSS can recommend the selection of a particular architecture style based on fuzzy logic and historical information of previous projects [131]. A hierarchy of software architecture metrics and a set of preference relations between them are the basis of another MCDM method for the selection of architectural patterns [139]. In the context of CBSE, BAREMO is a MCDM method that applies AHP to select software components from repositories based on four criteria, namely production time, cost, quality and risk [114].

1.2. Search techniques

1.2.1. Search and optimisation

The field of artificial intelligence (AI) is closely related to the decision-making process, since intelligent techniques can support some of its steps [60]. In this sense, an intelligent DSS is a special type of knowledge-based DSS that applies AI techniques to automate some tasks of the process, such as the analysis of information or the search of solutions [150]. In particular, Figure 1.1 depicts the correspondence between problem-solving and the common steps of the decision-making process [86]. In this context, a *search problem* is formulated in terms of *goals*, *states* and *actions* [166]. These elements represent the information that an intelligent technique

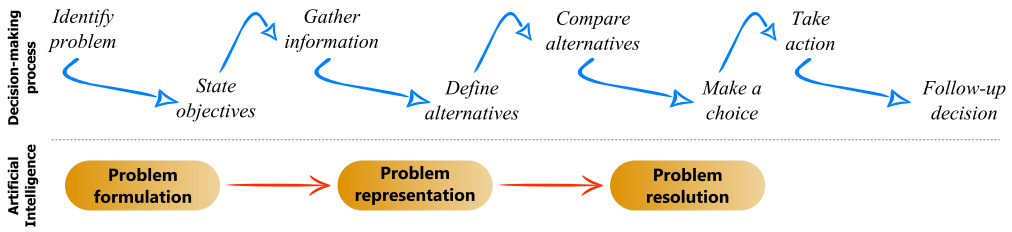


Figure 1.1: The role of artificial intelligence in the decision-making process

needs to make its decisions, i.e. an objective to be pursued, the possible *solutions* that can lead to the achievement of the goal, and a set of transitions to move from one solution to another. Once the problem has been defined, a *search algorithm* will perform as follows [166]:

> A **search algorithm** "takes a problem as input and returns a solution in the form of an action sequence (...), i.e. a path from the initial state to a state that satisfies the goal."

Traditional search algorithms translate this idea into building a tree of possible states and traversing through the tree in order to find the solution. In each step of the process, the algorithm applies an *operator* to expand the set of states to which it can move and chooses one to proceed with the search. If no additional information is used to choose the next state, the algorithm is performing a *blind or uninformed search*. Examples of this type of search are *breadth-first* and *depth-first* search. As opposed to blind search, heuristics perform a more *informed search* in the sense that they use problem knowledge to determine which state is the most promising to go next. Therefore, heuristics are specifically defined for the problem under study and can be highly effective to accelerate the search. However, heuristic algorithms partially explore the search space, meaning that finding the optimal solution cannot be guaranteed. Examples of heuristic algorithms are A^*, *branch and bound* methods, and *greedy* search.

Some search problems cannot be defined in terms of states and actions, but as a set of decision variables whose optimal values need to be determined. This type of search problem is known as *optimisation problem*, whose mathematical formulation requires three elements: 1) n *decision variables*, either discrete or continuous, whose values represent a solution; 2) problem constraints, expressed as inequality and/or

equality functions; and 3) an *evaluation function*, a.k.a. objective function, to map the solutions to a numeric value representing its quality. Under these assumptions, solving an optimisation problem can be stated as follows [42, 130]:

Given a search space Ω, solving an **optimisation problem** consists in finding the solution $x = (x_1, ..., x_n)$, that maximises (or minimises) the evaluation function $f : \Omega \subseteq \Re^n \rightarrow \Re$, subject to existing inequality (g) and equality (h) constraints:

$$\begin{aligned} f(x) &\geq f(y) \quad \forall y \in \Omega \\ g_i(x) &\leq 0 \ , \quad i = \{1, ..., m\} \\ h_j(x) &= 0 \ , \quad j = \{1, ..., o\} \end{aligned} \tag{1.1}$$

Regardless of how the search problem is formulated, search algorithms can be classified on the basis of four criteria [166]:

- *Completeness*, which guarantees that the algorithm can find a solution, if such a solution exists.
- *Time complexity*, which refers to the time needed to find a solution.
- *Space complexity*, which focuses on memory requirements during search.
- *Optimality*, which means that the algorithm is able to find the best solution.

1.2.2. Metaheuristics

The term *"meta-heuristic"* was introduced in 1986 to refer to a high level mechanism to escape from local optima [75]. A formal definition is given below [140]:

A **metaheuristic** is "an iterative generation process which guides a subordinate heuristic by combining intelligently different concepts for exploring and exploiting the search space, learning strategies are used to structure information in order to find efficiently near-optimal solutions".

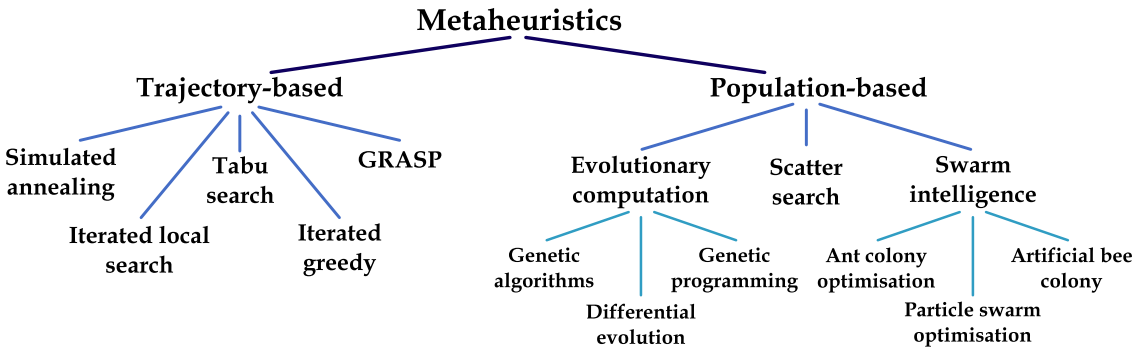


Figure 1.2: Classification of metaheuristic techniques

Metaheuristics have become highly popular due to their efficiency and adaptability. The success of metaheuristics strongly relies on achieving a good balance between *intensification* and *diversification* [26]. The former concept refers to the need of exploiting the knowledge acquired during the search, e.g. some properties of good solutions or their location in the search space. The latter term concerns the exploring capabilities of the method, which are essential to avoid local optima. Although there is a great variety of metaheuristic methods, most of them are characterised by the following properties [187]:

- *Iterative*, meaning that they start from complete solution(s), as opposed to greedy approaches that construct them from scratch.
- *Stochastic*, since they apply random rules in some steps of the search to escape from local optima.
- *Memory-based*, which refers to the use of information extracted from the solutions or the search process to enhance their search capabilities.
- *Bio-inspired*, as they simulate natural processes that exhibit intelligent behaviour.

A commonly accepted classification of metaheuristics is based on the number of solutions that are simultaneously handled [26, 27]. On the one hand, metaheuristics based on trajectory only maintain one solution that is iteratively improved. On the other hand, population-based metaheuristics manage a group of solutions in each step of the search. Population-based metaheuristics can be further divided

according to the biological processes in which they are inspired. In this sense, EC is based on the evolution of species, whereas swarm intelligence (SI) simulates the collective behaviour of different living beings like ants and birds. Figure 1.2 shows a classification of the most popular methods. Those applied in this Ph.D. Thesis are briefly explained in next sections.

Metaheuristics based on trajectory

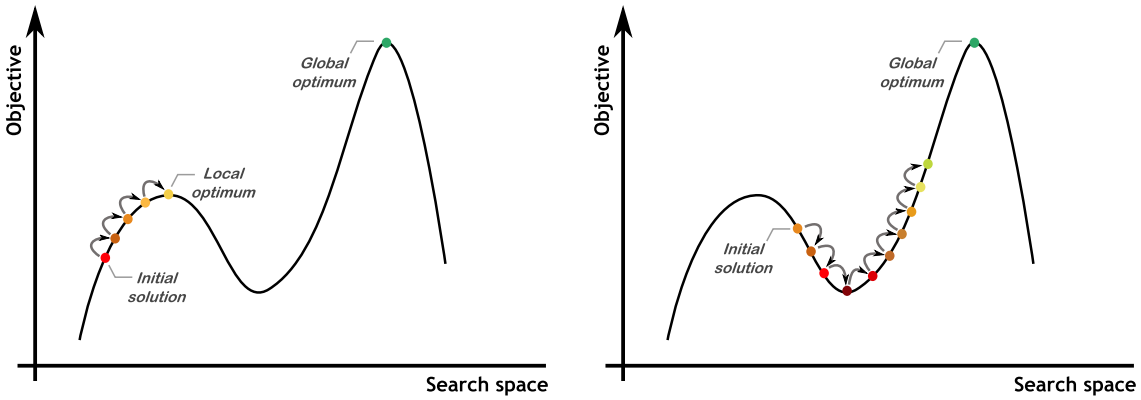
Trajectory-based metaheuristics start from a single solution and iteratively explore its *neighbourhood* looking for better solutions. Thus, the two basic elements of this type of technique are: 1) a method to generate neighbouring solutions, which depends on how the problem is encoded; and 2) a decision rule to accept one of these neighbours as the new solution. A formal definition of neighbourhood is provided next [187]:

> "In a discrete optimisation problem, the **neighbourhood** $N(s)$ of a solution s is represented by the set $\{s' \mid d(s', s) \leq \epsilon\}$, where d represents a given distance that is related to the move operator".

The simplest approach is to follow a steepest-ascent[1] strategy, known as hill climbing (HC) [166] (see Figure 1.3a). HC generates a set of neighbours, and the best one according to the evaluation function is chosen for the next iteration. HC is a local search (LS) method that can rapidly progress towards an optimum, but it could also be easily trapped into a local optimum. To avoid this, multiple runs can be performed, each one starting from a different solution. Due to the lack of randomness and the absence of memory structures, HC is not usually considered a metaheuristic as such, but represents the baseline approach for defining more advanced techniques.

A first method to overcome the limitations of HC is simulated annealing (SA) [102], which takes its name from the physical annealing process used in metallurgy. In this process, metals undergo fast heating and are then slowly cooled to reach an appropriate energy state [187]. Analogously, this final state corresponds with the global optimum in SA, and the energy is determined by the evaluation function. The key characteristic of SA is that it can accept movements to worse neighbours

[1] Steepest-descent for minimisation problems.



(a) Hill climbing

(b) Simulated annealing

Figure 1.3: Examples of trajectory-based search methods

based on a probability that decreases with the elapse of the search. Figure 1.3b illustrates this process. The probabilistic acceptance rule depends on the amount of objective degradation and the *temperature* parameter, which is modified according to a cooling scheme.

Finally, tabu search (TS) [75] differs from SA in that it adopts a memory mechanism to avoid local optima. More specifically, TS allows movements to worse solutions if no neighbour improves the current one, but maintains a *tabu list* to prevent acceptance of previously visited solutions [74]. This short-term memory is updated every iteration, usually storing a fixed number of accepted moves or their characteristics. TS can be enhanced with the combination of medium-term and long-term lists to promote intensification and diversification, respectively [187].

Metaheuristics based on populations

EC is based on the Darwinian principles of natural evolution, such as the survival of the fittest, and is probably the most popular population-based metaheuristic [62]. In EC, candidate solutions are called *individuals*, which are characterised by a *phenotype*, i.e. the real-world solution, and a *genotype*, i.e. its computational encoding. The genotype is usually a linear structure, e.g. an array, containing *genes* that store the values of the decision variables. Following with the simile, the evaluation function is here called *fitness* function, since it provides a value of the adaptation of the individual that can be used to compare it against others.

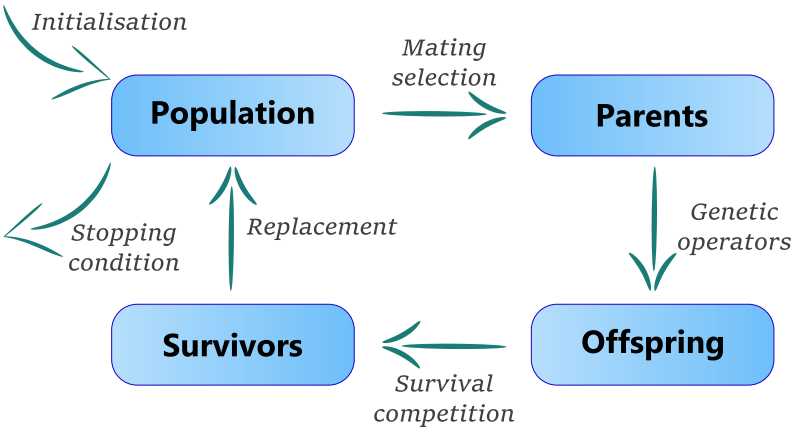


Figure 1.4: The generational cycle of an evolutionary algorithm (adapted from Eiben and Smith [62])

Every evolutionary algorithm (EA) follows an iterative process similar to the one shown in Figure 1.4. As can be seen, the evolutionary process starts with the random creation of a population of individuals, which are evaluated by the *fitness* function. A selection process, which is often based on the quality of the individuals, picks some of them to become *parents*. Parents are then *recombined* to generate new solutions by interchanging their genetic information. The descendants can be also *mutated*, which consists in applying small alterations in their genotypes. Often, each genetic operator is applied with a configured probability. The resulting *offspring* will compete against current population members to become part of the next population, a process that can also be based on their fitness values. This process is repeated for a number of *generations*, i.e. iterations, until a stopping condition is met. The application of selection pressure, elitism and genetic operators provide a proper balance between diversification, which is preferred in the first generations, and intensification, which is required in the last part of the process.

EAs can be used in combination with other metaheuristics or heuristics in order to exploit their respective abilities. In general, the idea is to incorporate additional knowledge in some steps of the evolutionary search [62]. For instance, random initialisation can be replaced by some heuristic procedure, while problem-specific information can be considered within the genetic operators in order to perform more informed transformations. A memetic algorithm (MA) is an example of hybrid technique [25] that integrates local improvements into the evolution by means of methods like HC or SA. The design of a MA involves deciding the step of the

evolution in which LS is applied and what individuals will be the initial solutions, among other aspects [108].

1.2.3. Optimisation with multiple objectives

Real-world decision scenarios often present multiple objectives that should be simultaneously considered [51]. Furthermore, these objectives are usually in conflict, meaning that it is not possible to find a solution achieving optimal values for all objectives. The field of MOO is focused on solving this type of problem, known as multi-objective problem (MOP). MOO is strongly related to MCDM, since its main goal is to find a number of solutions representing different trade-offs among the objectives, so that a *decision maker* can choose the solution to be implemented based on additional preferences [51]. The mathematical formulation of a MOP only differs from Equation 1.1 in that the goal is to maximise (or minimise) k objective functions [42]:

> Given a **search space** Ω, solving an **MOP** consists in finding the solution $x = (x_1, ..., x_n) \in \Omega$, that minimises (or maximises) the evaluation function $F(x) = \{f_1(x), ..., f_k(x)\ ,\ k \geq 2\}$, subject to existing constraints.

Due to the presence of several objectives, a new concept of "optimality" is required to specify whether one solution is better than another. In MOO, the Pareto dominance principle establishes that Pareto optimal solutions are those solutions that are *non-inferior* than any other solution [42]:

> Given a maximisation MOP, a solution x **dominates** another solution y ($x \succ y$) if and only if $f_i(x) \geq f_i(y) \wedge \exists j \mid f_j(x) > f_j(y)\ ,\ i = \{1, ..., k\}$

The entire set of Pareto optimal solutions is referred as the Pareto set (PS). Figure 1.5 illustrates the concept of Pareto dominance and the possible relationships between solutions. The set of all non-dominated vectors comprises the Pareto front (PF). Based on these concepts, the goals in MOO are [51]:

- To find a good approximation to the optimal PF (a.k.a. PF_{true}).

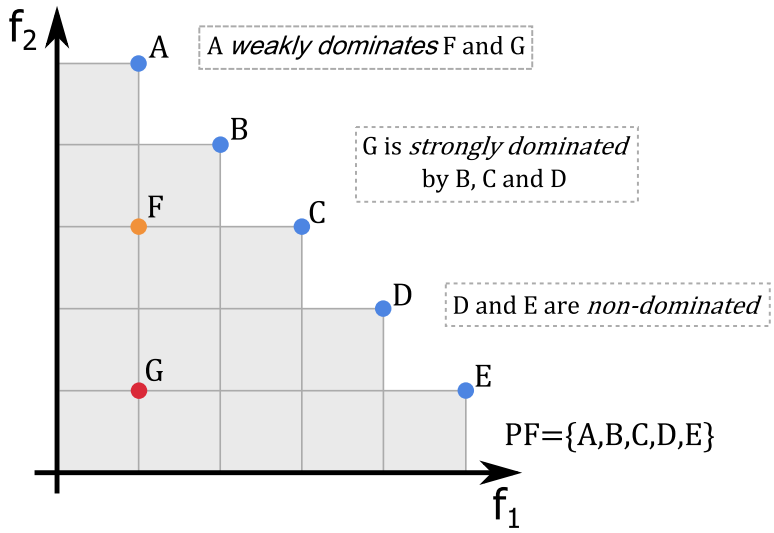


Figure 1.5: The concept of Pareto dominance in multi-objective optimisation

- To find a diverse and well-spread set of solutions.

Quality indicators are performance measures that can be used to numerically assess the quality of the returned PF, or PF_{known}, according to these aspects [42]. More specifically, they usually evaluate some of the following properties [210]: 1) convergence, i.e. the distance to the optimal PF; 2) uniformity, which ensures a good distribution of the solutions; and 3) spread, which measures the extent of the PF. In addition, quality indicators can be classified according to the number of PFs required for its computation. In this sense, *unary* indicators estimate the quality of a PF only based on its vectors, whilst *binary* indicators require an additional PF, i.e. a reference PF, to return a relative value. Table 1.1 provides a list of the most used quality indicators, their description and properties [42, 161]. One of the most popular is hypervolume, whose value would correspond to the shaded area in Figure 1.5. The choice of a particular quality indicator to compare algorithm performance is not straightforward, so a general recommendation is to report several indicators [161].

Multi-objective evolutionary algorithms

An EA that has been adapted to deal with multiple objective functions is referred as a multi-objective evolutionary algorithm (MOEA) [209]. In general, a MOEA introduces some changes in the selection and replacement procedures, since these steps

Table 1.1: Quality indicators frequently used in MOO and their properties (C: coverage, U: uniformity, S: spread)

	Indicator	Definition	C	U	S
Unary	Hypervolume	Hyperarea covered by a PF	✓	✓	✓
	Spacing	Distance variance of neighbouring vectors		✓	
	Spread	Extent of the PF with respect to extreme vectors			✓
	Generalised Spread	Extension of spread for more than two objectives			✓
Binary	I_ϵ	Minimum shifting in one PF to dominate another	✓		
	Generational Distance	Shortest distance from PF_{known} to PF_{true}	✓		
	Inverted Gener. Dist.	Shortest distance from PF_{true} to PF_{known}	✓	✓	✓
	Maximum PF Error	Largest minimum distance to PF_{true}	✓		
	Coverage	Ratio of dominated vectors in a PF by another PF	✓		

often perform comparisons based on solution quality. More specifically, three evolutionary aspects need to be revisited [211]: fitness assignment, diversity preservation and elitism. Frequently, MOEAs define a function based on dominance criteria that allow ranking solutions according to their optimality. Also, specific mechanisms based on density estimation are used to maintain diversity. Elitism can be promoted by means of an external archive that stores the best solutions found so far.

Since the appearance of the first MOEA in 1985 [170], a plethora of algorithms have been proposed. The *first generation* of MOEAs is founded on the Pareto dominance, also including some niching or fitness sharing techniques [41]. The design of an elitism mechanism, either based on external archives or specific selection procedures, is the distinctive characteristic of the so-called *second generation* [41]. Two popular algorithms belonging to this category are SPEA2 (*Strength Pareto Evolutionary Algorithm 2*) [212] and NSGA-II (*Non-dominated Sorting Genetic Algorithm II*) [54], which are actually improved versions of first-generation algorithms. On the one hand, SPEA2 defines a fitness assignment method to determine a strength value for each individual, counting the number of solutions it dominates. It also applies a density estimation strategy based on clustering to select more diverse individuals and maintains a fixed-size archive. On the other hand, NSGA-II defines a sorting method to rank the solutions in fronts according to the dominance between them. A crowding distance is also computed to act as a secondary criterion when two solutions are in the same front. As opposed to SPEA2, NSGA-II does not use an archive, since the sorting method always guarantees that best solutions are kept.

Many-objective optimisation

Recently, the resolution of optimisation problems with a large number of objectives has gained increasing attention in MOO from both theoretical and practical views [110, 154, 155]. According to recent literature, a many-objective problem (MaOP) requires the definition of four objectives at least [35, 112]. First studies within many-objective optimisation (MaOO) reveal that traditional MOEAs suffer performance degradation as the number of objectives increases [101, 152, 171]. In an attempt to solve this, existing MOEAs were adapted to improve their performance in this new scenario, specially regarding diversity preservation [3, 106]. As the field has been studied in more depth, a number of challenges have been identified [35, 112]:

- Deterioration of selection pressure due to the exponential growth of the number of non-dominated solutions.
- Inefficiency of crossover operators, since solutions tend to be far from each other.
- Some search procedures become computationally expensive, e.g. hypervolume calculation or distance-based strategies.
- Accurate representations of the optimal PF need to be comprised of a high number of solutions.
- Visualisation of high-dimensional PFs, which is essential for decision-making, requires the use of specialised techniques [199].

The growing interest in solving MaOPs has led to the appearance of a new type of specialised algorithm, referred as many-objective evolutionary algorithm (MaOEA). MaOEAs are often categorised into *families*, depending on the mechanisms proposed to face some of the aforementioned challenges [110, 196, 198]. Table 1.2 provides a compilation of current approaches and a list of some representative algorithms. The algorithms applied in this Ph.D. Thesis are presented next, grouped by family.

Firstly, two algorithms that makes use of relaxed dominance principles are introduced. ϵ-MOEA [53] divides the objective space into fixed-length hypercubes to define a new dominance relation, named ϵ-dominance, over the resulting landscape.

Table 1.2: Families of many-objective evolutionary algorithms

Family	**Description**	**Algorithms**
Relaxed dominance	Relaxed forms of dominance are considered to increase selection pressure.	ϵ-MOEA, MDMOEA, GrEA
Diversity	New mechanisms to mitigate the impact of previous diversity preservation techniques.	SPEA2+SDE, VaEA, NSGA-II+SDE
Decomposition	Objective values are aggregated to evaluate or compare solutions.	MOEA/D, MSOP, MOEA/DD
Indicator	An indicator evaluates the contribution of each solution to the quality of the PF.	HypE, SMS-EMOA, IBEA, MOMBI
Reference set	Reference points, configured by the user or automatically generated, guide the search.	NSGA-III, RVEA
Preferences	The search is directed towards the region of the PF that represent user's preferences.	R-NSGA-II, PBEA
Dimensionality reduction	Redundant objectives are excluded from the search process.	PCA-NSGA-II, PCSEA

Thus, solution x ϵ-dominates solution y if x belongs to better or equal hypercubes for all the objectives and to a better hypercube for at least one objective than y. In each iteration, ϵ-MOEA selects one parent from the current population and another one from an archive of solutions. Offspring will survive depending on the hypercubes to which they belong and those already filled by archive members. Another relevant algorithm is GrEA (Grid-based Evolutionary Algorithm) [205]. It also relies on the notion of landscape partition, though hypercubes – here called grids – are dynamically created. Initially, GrEA uses grid information to choose the most promising individuals for reproduction. Inspired by NSGA-II, GrEA also ranks the population by fronts during replacement, but substitutes the crowding distance by novel grid-based metrics.

The first indicator-based approach is IBEA (Indicator-based Evolutionary Algorithm) [213], which defines a general evolutionary algorithm where any quality indicator could be used as a fitness function. The authors show the applicability of the approach in combination with two indicators, i.e. hypervolume and I_ϵ. Binary tournaments based on the fitness function are performed to select parents. During replacement, the worst individuals with respect to the fitness function are discarded. Another indicator-based algorithm is HypE (Hypervolume Estimation Algorithm) [13], which estimates the portion of hypervolume that can be attributed

to each individual. This value is used to determine the winner of binary tournaments during parent selection, while those solutions that contribute the least to overall hypervolume are removed from the population.

Focusing on decomposition techniques, MOEA/D (Multiobjective Evolutionary Algorithm based on Decomposition) [207] divides the original problem into several subproblems to be simultaneously optimised. More specifically, each individual is associated to one subproblem by means of a weighted vector. Neighbourhood information based on these vectors is considered during replacement, so that each offspring is matched to the subproblem it can best solve. This algorithm uses an archive to keep all non-dominated solutions found throughout the evolution process. Lastly, NSGA-III [52], the improved version of NSGA-II for solving MaOPs, requires a set of reference points to guide the search. Parents are randomly selected before applying genetic operators. Then, individuals are associated to their closer point, and one individual per reference point is selected for survival thus promoting diversity.

1.2.4. Interactive optimisation

Decision-making is characterised by uncertainty factors, often requiring a progressive understanding of causes, alternatives and consequences [146]. The potential of AI can be insufficient when decision variables or objectives cannot be formally defined a priori. Other factors hampering the effectiveness of AI techniques conceived for decision support are the simplification of the real-world problem and user's reticence to trust and accept automatically generated results [129]. In order to build an intelligent DSS of real practice in complex decision scenarios, humans have to be involved in the process [17]. Cooperation between domain experts and the intelligent DSS is crucial to improve the efficiency of AI techniques and to ensure that both the process and the outcomes meet user's expectations.

The need of putting the "human-in-the-loop" has been also acknowledged by the EC community, giving rise to the field of interactive evolutionary computation (IEC) [186]. Originally, IEC was conceived as a way of substituting the fitness function by subjective human evaluation for those creative tasks in which defining an accurate fitness function is not possible. Within MOO, user's preferences –

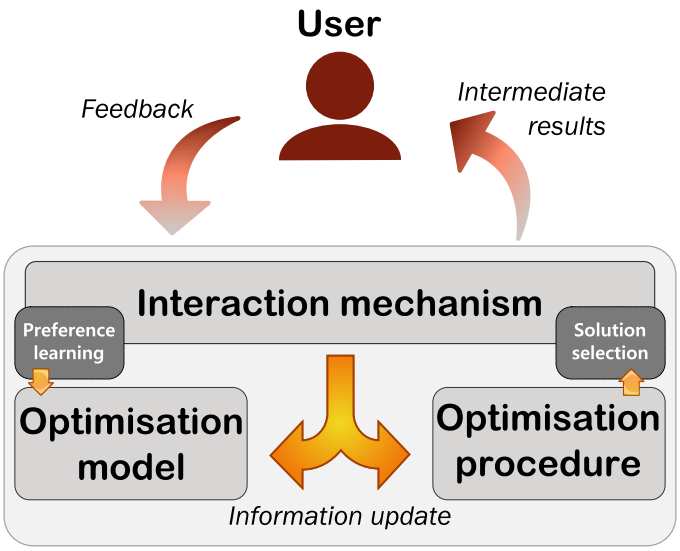


Figure 1.6: Overview of an interactive algorithm (adapted from Meignan *et al.* [129])

usually in form of aspiration levels for each objective – can be interactively specified so that the search is directed towards certain regions of the PF [31]. More generally, *interactive optimisation* encompasses any optimisation method, including metaheuristics, in which the human actively participates in any step of the process to provide feedback [129].

The principal components of an interactive optimisation algorithm are depicted in Figure 1.6. As can be seen, the role of the human consists in providing his/her opinion in light of intermediate results generated by the algorithm, e.g. some candidate solutions shown for human evaluation. The feedback can be integrated into the optimisation model to redefine some objectives or constraints. A model of user's preferences could also be *learned* in order to extract knowledge and reuse it in other contexts. In addition, the gathered information can have direct impact on some components of the algorithm, e.g. manually modifying solutions. The taxonomy proposed by Meignan *et al.* [129] provides a framework to characterise these and other relevant factors, such as the information lifetime and type of feedback integration. Several authors have also proposed specific techniques to choose representative solutions for presentation [175] and schedule interactions [125]. Lastly, some implementation issues should be also considered in order to offer an engaging user experience, e.g. intuitive and easy-to-use interaction interfaces [174]. All these aspects are essential to prevent the fatigue experienced by the user due to frequent

interaction and information overload, which can lead to some inconsistencies and loss of interest.

1.3. Search-based software engineering

1.3.1. Origin and characteristics

Although AI and SE have been traditionally viewed as separate disciplines, several authors have advocated for potential ways of collaboration between them. As pointed out by D. Partridge in 1988 [147]: "there is opportunity for the use of AI in system development and maintenance environments to support the development of conventional software. Constructing software is a real intelligence task". More specifically, the "AI perspective" of SE was conceived as the "reformulation of SE processes in AI terms and an attempt to solve them entirely within AI" [9]. However, other authors have argued that some AI-based support to software construction is desirable, but the human still has to play a central role in the process [147]. As AI has gained maturity, its methods have been progressively adopted by the SE community, giving rise to the following interdisciplinary, and sometimes overlapped, research areas [78]:

- Machine learning for SE, which is focused on building classification or predictive models for effort estimation [202] or fault prediction [119].
- Data mining for SE, which extracts knowledge from historical data, including *specification mining* [8] and *mining software repositories* [95].
- Probabilistic SE, which mainly refers to the application of Bayesian networks to model quality aspects [189] or analyse requirements [55].
- SE guided by search, which proposes the reformulation of SE activities as optimisation problems to be solved by means of search techniques [79].

Initial attempts to apply search algorithms to SE activities date back to the late 1990s, with works focused on testing, cost estimation and automatic programming [81]. However, the idea of applying search techniques to solve optimisation problems within SE was formalised as research field by M. Harman and F. Jones

in 2001 [81]. Since then, SBSE has experienced a rapid development, with a wide range of application domains practically covering the entire software development process [79]. According to Harman and Jones [81], SE activities present a number of characteristics that make them suitable for optimisation, such as the presence of competing objectives, the existence of potential solutions, a need to sort out the good solutions from the bad, and the lack of precise rules for finding these solutions. Under these assumptions, SBSE proposes that [40, 81]:

> SE tasks can be reformulated as optimisation problems, for which two elements need to be defined: 1) a computational representation for possible solutions and 2) a fitness function defined in terms of the selected representation.

Compared to other engineering disciplines, the intangible nature of software artefacts make relatively easy to capture their characteristics as part of a computational encoding. Furthermore, software metrics, which have been extensively investigated by the SE community, represent ideal candidates for the definition of fitness functions [80]. Similarly, the idea of achieving trade-offs, e.g. cost vs. time or cohesion vs. coupling, perfectly matches with the precepts of MOO, whose techniques have been explored more recently [169]. For those situations in which more qualitative aspects need to be considered [168], SBSE can also benefit from interactive approaches [179]. In fact, the potential of SBSE goes beyond the automation of laborious tasks currently done by engineers, since it also represents an opportunity to generate unexpected solutions, provide insights into software quality measurement and offer innovative tools to practitioners [78, 81]. All these aspects make SBSE well-suited for decision support in SE, especially in the earliest stages of the development process [84].

Due to the broad scope and exponential growth of the field, research within SBSE is usually reviewed with respect to the phase of the software lifecycle in which the problem is defined. In this sense, SBSE approaches for project management are focused on automatic generation of estimation models and staff schedules [68]. For requirements elicitation, SBSE can be applied to select and prioritise requirements [151]. Both object-oriented and service-oriented design paradigms present

problems for which search techniques can be helpful [158], such as class responsibility assignment or service composition, respectively. Automatic programming and code improvement are areas related to SBSE too [149]. Search-based software testing is the most prolific area, with research into test data generation and test case selection, among others [127]. Refactoring and modularisation are recurrent topics within search-based maintenance [124, 137]. Finally, it should be noted that SBSE has also found synergies with particular SE paradigms that span across multiple phases, such as cloud computing [82], software product line (SPL) [111] and model-driven engineering (MDE) [28].

1.3.2. Search-based software design

The area of search-based software design (SBSD) has experienced a significant growth in the last years [158], where any form of software design, from object-oriented to service-oriented, can be studied. Even so, the creativity and intuition required in many design tasks make SBSD particularly challenging. In this sense, the suitability of interactive optimisation has been highlighted [84], and several works integrating user's preferences can be found in the literature [67]. In addition, the wide range of available modelling techniques and the abstract nature of their artefacts imply that problem encoding, fitness evaluation and solution transformation in SBSD are often problem-specific and hard to define [158]. In fact, how to automatically assess design quality beyond structural aspects related to cohesion and coupling is yet an open issue in SBSD [135, 158]. What also remains less explored is how design choices can influence later stages of the development process, such as testing or maintenance, and whether such an impact could serve as an evaluation of design adequacy [83]. Other identified gaps are the support to early architectural design and the integration of good practices like design patterns [158].

Despite this, SBSD literature now covers a broad range of problems and techniques. Focusing on object-oriented design, one of the first and most studied problems is class responsibility assignment [30]. Here, the objective is to find the optimal allocation of attributes and operations for a given class diagram. Therefore, the evolutionary algorithm has to move attributes and operations among classes guided by cohesion and coupling criteria. In his Ph.D. Thesis, C. Simons covered the early conceptual design, for which attributes and operations extracted from use cases

were combined to create a class diagram [178]. His works [180–182] are particularly relevant within SBSD as they propose human-in-the-loop approaches that combine structural metrics with more subjective criteria like elegance. Furthermore, one of these works applies ant colony optimisation (ACO) as search technique [182], as opposed to most of the SBSD studies that frequently adopt EAs [158]. Refactoring at design level has also been studied [85, 120]. Harman and Tratt explore the use of MOO techniques to find the best sequence of "move method" transformations for an input class diagram [85]. More recently, Mansoor *et al.* have also adopted a MOO approach to propose a multi-view approach, in which both UML class and activity diagrams are considered [120]. In addition, they implement an extensive catalogue of refactoring operations, including pushing down and pulling up properties and operations along class hierarchies.

Several design tasks in the context of MDE have been recently reformulated as optimisation problems. Firstly, model transformation is redefined as the problem of automatically deriving transformation rules from a base of examples [99]. In this case, particle swarm optimisation (PSO) and SA are the search techniques applied to transform UML class diagrams into relational schemas. For model merging, a genetic algorithm can be applied to find a tentative combined model in which the number of invalidated operations due to existing conflicts is minimised [100]. In fact, this process is equivalent to refactoring, since the merged model is produced as a sequence that integrates operations belonging to class models developed in parallel. Another proposal within search-based MDE deals with metamodel matching, in which the goal is to find the optimal correspondence between metamodel elements [98]. Here, a genetic algorithm seeks for good solutions that are then refined by using SA. Very recently, modularisation of model transformation programs has been formulated as a many-objective problem [69], NSGA-III being the selected algorithm to solve it.

Focusing on service-oriented computing, web service composition is a common practice to build software applications from third-party services, thus reusing functionality [176]. Problems related to web service composition have been also subject of research within SBSE. The automatic binding of candidate web services for a given composition structure was introduced by Canfora *et al.* in 2005 [34], a problem known as QoS-aware web service composition (QoSWSC). They proposed a single-objective formulation in which several quality-of-service (QoS) properties, such as

cost and time, are aggregated into a fitness function. Since then, diverse EAs have been proposed to address this problem, though the suitability of other techniques, such as GRASP [145], has been also explored. A recent trend is to consider each QoS property as an independent objective and apply MOEAs [45]. Many-objective formulations, ranging from 5 to 10 objectives, have also appeared [50, 191, 197, 206]. Another problem related to service composition concerns how services are orchestrated, i.e. which is the best composition workflow. Genetic programming, a variant of EC specially well-suited to evolve tree structures, can be applied to address this problem [163]. Finally, approaches combining both service selection and workflow construction can be found in recent literature [48, 65].

1.3.3. Software architecture optimisation

Software architecture optimisation methods seek for the automatic specification and improvement of architectural models [4]. More specifically, optimisation methods have been mostly used to arrange elements of an architectural specification or semi-automatically derive new models according to predefined quality attributes and other existing constraints. The taxonomy proposed by Aleti *et al.* classifies existing approaches according to the following aspects [4]: *problem*, which covers the system domain, existing constraints and quality attributes under study; *solution*, which includes solution encoding, type of transformations and optimisation strategy; and *validation* in terms of practicality and algorithm performance. In what follows, state-of-the-art SBSE methods applied to architecture optimisation, grouped by the optimisation goal they pursue, are described.

Architecture synthesis from use cases has been addressed with EAs [157]. In this problem, the algorithm produces a low-level architectural specification composed by classes and interfaces, which also integrates design patterns. In their works, Räihä *et al.* proposed hybrid approaches and specific operators to enhance the evolutionary search [160, 177]. Furthermore, they later explored the suitability of multi-objective [159] and interactive [192] approaches. Also starting from use case modelling, the identification of logical components as groups of related use cases has been recently proposed as an optimisation problem [87]. In this method, hierarchical components are created by an EA guided by a cohesion measure. In a later work [88], analysis classes derived from use cases are the inputs of the evolutionary search, so

each component is created from one "key" class and its related classes. In addition, the authors included architectural preferences and constraints in the search model, such as the presence of legacy systems.

Software architectures can be constructed following an assembling approach too, for which black-box components, e.g. COTS, need to be selected and integrated. This type of design process presents additional optimisation opportunities. For instance, Baker *et al.* have proposed the use of SA and greedy algorithms to select the subset of pre-existing components to be included in the next release of a system [14]. Support to "buy or build" decisions when designing a component-based architecture was considered based on cost and reliability criteria [44]. More recently, Vescan and Şerban have presented an evolutionary approach to automatically generate a multilevel architecture by selecting components from repositories [195].

There are several studies applying search techniques to automatically recover components from source code or their abstracted dependency graphs. In both cases, the existence of an already developed system is assumed, so the process is carried out for re-engineering purposes. A first approach can be attributed to R. Lutz, who proposed a genetic algorithm for evolving tree structures representing hierarchical module decompositions [116]. Hierarchical clustering is the technique chosen by Maqbool and Babri [121]. They studied the adequacy of several similarity measures for grouping related functions, which are considered the basic entities of legacy systems. Similarly, Cui and Chae [47] compared the performance of several hierarchical clustering methods for component identification from legacy systems. ROMANTIC is an optimisation approach that looks for the recovery of an intentional component-based architecture from object-oriented code [38]. Both SA [36] and EC [173] were used as part of the recovery process. The approach has been extended to include clustering methods, a semantic-correctness measurement model and other sources of information, such as UML diagrams and logs from control version systems [37, 97].

The design of a software architecture in compliance with architectural styles and patterns has been scarcely investigated. Firstly, a coevolutionary approach for architectural synthesis combines responsibility allocation and preservation of pattern constraints [204]. More specifically, the authors defined some metrics to numerically assess conformance to the MVC pattern. Recovering a layered architecture from a package dependency model was recently formulated as an optimisation problem [20].

Similar to the previous approach, a set of constraints defining the essence of the style are included in the recovery process, which is based on TS. Within the context of SPL, Mariani *et al.* have proposed specific operators to preserve the architectural style of a product line architecture, represented as a UML class diagram, during its improvement [123]. These operators are conceived for two particular styles, namely layered architecture and client/server architecture.

Some other works study how SBSE can help improving some aspects of existing architectural models, sometimes referred as architecture refactoring. Herold and Mair [90] proposed a recommendation approach, based on HC, to find a sequence of "move class" refactoring operations, thus suggesting how to repair architecture violations due to erosion. The aim of a another search-based refactoring approach is to reduce architectural bad smells in component-based systems [96]. The method combines detection rules based on metrics with an EA, which takes a source code model as input.

Architecture optimisation during deployment has been extensively studied, often relying on MOO techniques. Two recurrent topics are component reconfiguration and allocation to physical nodes. In both cases, components are viewed as black-box artefacts with annotated quality attributes. Focusing on reconfiguration, L. Grunske proposed a MOEA to decide which components should be replicated based on a cost/reliability trade-off analysis [77]. Other reconfiguration approach focuses on finding one or more compatible components to substitute an obsolete one [56]. In this case, some heuristics guide a branch and bound strategy to re-build the system. Focusing on allocation problems, the component deployment problem consists in finding the optimal assignment of components into available hosts to guarantee proper reliability. NSGA-II was embedded into a tool called ArcheOpterix to solve this problem [128]. A similar tool is PerOPTeryx, which can simultaneously tackle component allocation, server configuration and component selection [107]. The tool combines an analytic process with NSGA-II in order to improve an initial architecture specification in terms of performance and availability. Similarly, AQOSA toolkit integrates popular MOEAs (NSGA-II, SPEA2 and SMS-EMOA) to optimise resource utilisation, cost and latency [63]. Here, the objective is to determine an optimal hardware topology, also considering component allocation and replacement. DeSi is another tool that can support the analysis and improvement of the current

deployment architecture of a distributed system via greedy and evolutionary algorithms [118]. The optimisation process is guided by a single fitness function that aggregates multiple QoS properties, such as availability, latency or energy consumption.

Finally, two optimisation-based tools provide architects the necessary support to carry out additional tasks during the architecting process. On the one hand, ArchiTech is an ontology-based DSS for the management of architectural knowledge that uses SA as inference mechanism [7]. The system allows architects to understand the impact of their architectural decisions, e.g. architectural style or development technologies, in several non-functional requirements, as well as to discover possible incompatibilities with previous decisions. On the other hand, SADHelper provides guidance on the production of wiki-based architectural documentation [133]. Based on a cost-effective analysis of stakeholders' information needs, documentation update plans can be suggested running NSGA-II.

2

Motivation and objectives

"A human must turn information into intelligence. We've tended to forget that no computer will ever ask a new question".
Grace Hooper

The analysis of the state of the art has led to the identification of research opportunities within the field of SBSD. More specifically, the area of software architecture optimisation lacks on methods to support architects on the identification of the functional blocks of the software during early software conception. This type of analysis would allow architects to discover a candidate system structure, thus making its analysis and improvement easier. From the point of view of search techniques, it seems appropriate to study the applicability of advanced techniques, such as many-objective optimisation and interactive optimisation, for the discovery of software architectures. On the one hand, SE problems like this one often present a high number of factors to be considered, though they are hard to be defined as objective functions. If such objectives are properly formulated, then the performance of novel many-objective search techniques could be analysed. On the other hand, potential ways of collaboration between the engineer and the search algorithm are yet to be explored in SBSD. In this sense, it is necessary to carry out a in-depth study of the interaction requirements, as well as to determine the most appropriate

mechanisms to support it. It would allow the development of an effective interactive method combining the abilities of the involved actors. The design of experiments to validate the proposed approaches would also provide insights on the strengths and weaknesses of these techniques, thus contributing to promote their broader adoption within SBSE.

In this chapter, the purpose of this research is described in terms of a general objective and its specific subobjectives. For each of them, at least one research question (RQ) is formulated. RQs are equivalent to hypotheses since they both represent a formal statement of the phenomenon or concept under study [46]. The difference lies on the intention of such statement, since hypotheses make predictions about the expected results based on the relation between dependent and independent variables, whereas RQs are focused on the exploration of the factors surrounding the phenomenon with the aim of deriving new knowledge or improving previous solutions. Therefore, RQs are more appropriate when the nature of the research does not allow making predictions, which is often the case of solution-oriented qualitative research [46]. The identification of RQs is a well-established practice in SE research [61], which has been also adopted by the SBSE community. This chapter ends with a brief summary of the research tasks carried out to fulfil the objectives and the results derived from them, which serves to establish a direct correspondence between objectives and publications.

2.1. Objectives

The general objective of this Ph.D. Thesis is stated as follows:

> *Development of search models based on metaheuristic algorithms to provide semi-automatic support to software engineers in early stages of the software development process.*

This general objective was divided into the following subobjectives:

- **O_1:** Analysis of the state of the art in SBSE, especially in the areas of SBSD and architecture optimisation, to identify open problems and the techniques that could be applied to their resolution.
- **O_2:** Design and development of a metaheuristic model based on EC to support the discovery of component-based software architectures, including the study of the problem characteristics (analysis information, quality metrics, expert's needs, etc.)
- **O_3:** Design and development of advanced search models, including multi-objective approaches and hybrid techniques, to study their suitability to the decision support process. Study of their applicability to different domains within SBSD.
- **O_4:** Design and development of an interactive approach to incorporate human expertise in the context of software architecture optimisation. Comparison with previous approaches.

2.2. Research questions

In order to address O_1, an extensive literature review needs to be carried out. The following RQs are proposed to guide such an analysis:

- **RQ_1:** *What are the current gaps within SBSD and, more specifically, software architecture optimisation?*
- **RQ_2:** *How have SBSE studies applied multi- and many-objective optimisation techniques?*
- **RQ_3:** *In what ways has interactivity been adopted within SBSE?*

Firstly, RQ_1 would allow studying the state of the art regarding the application of search techniques to support the software design process. Within this area, new challenges in software architecture optimisation could be identified. Regarding the search approaches, RQ_2 would serve to analyse which are the current techniques being used, with a special focus on those following the principles of MOO and

MaOO. Finally, $\mathbf{RQ}_3$ would provide a deeper understanding of the potential of interactive optimisation, whose characteristics seem to fit with the requirements of SBSE.

Focusing on objective $\mathbf{O}_2$, two RQs have been defined:

- $\mathbf{RQ}_4$**:** *Can single-objective EAs help the software engineer to identify an initial candidate architecture of a system at a high level of abstraction?*
- $\mathbf{RQ}_5$**:** *How does the configuration of the algorithm influence both the evolutionary performance and the quality of the returned solution?*

The aim of $\mathbf{RQ}_4$ is to explore the use of EC as a supporting method to find optimal software architectures. More specifically, the characteristics of the architecture discovery process should be analysed in order to design an effective EA. Due to the lack of previous studies focused on this problem, the experimental evaluation of the evolutionary model should be oriented towards the study of the influence of its parameters, as expressed in $\mathbf{RQ}_5$.

The knowledge acquired from this first approximation to the problem could indicate the need of additional design evaluation criteria, which would demand the application of more advanced techniques. Accordingly, the following RQs are formulated for objective $\mathbf{O}_3$:

- $\mathbf{RQ}_6$**:** *How do a larger number and combination of metrics influence the search-based discovery of software architectures?*
- $\mathbf{RQ}_7$**:** *How do state-of-the-art MOEAs and MaOEAs perform when multiple objectives guide the discovery process?*
- $\mathbf{RQ}_8$**:** *How can local search be effectively integrated in the multi-objective evolutionary discovery of software architectures?*

Firstly, $\mathbf{RQ}_6$ emphasises the need of including more decision factors in the discovery process, as a way to adapt it to particular requirements of the system. Additional software metrics reflecting diverse quality properties could be considered as independent objective functions to be optimised. If a large number of conflicting

metrics is required, then the performance of MOEAs can be greatly degraded and MaOEAs should be considered instead. With this purpose in mind, $\mathbf{RQ}_7$ is posed. In addition, the performance of *pure* EAs can be enhanced with trajectory-based metaheuristics, a process that largely depends on the particularities of the problem under study. Therefore, $\mathbf{RQ}_7$ would serve to analyse their effect in the discovery of software architectures.

The last two RQs are stated in the context of objective $\mathbf{O}_4$:

- **$\mathbf{RQ}_9$:** *How can the qualitative judgement of the engineer be integrated into the evolutionary discovery of software architectures?*
- **$\mathbf{RQ}_{10}$:** *Does putting the human in the loop involve a significant improvement compared with not considering him/her along the optimisation process?*

Apart from the quantitative metrics that can be included in the problem formulation to measure different aspect of design quality, qualitative aspects could be required as well. To explore this complementary view, $\mathbf{RQ}_9$ seeks for the study of the role of the software engineer in the process. Once an appropriate interactive model has been designed, it will be possible to analyse the differences between interactive and non-interactive approaches as a way to reveal the benefits of the engineer's participation. This is the aim underlying the formulation of $\mathbf{RQ}_{10}$.

2.3. Relation between objectives and publications

This section details the research tasks that have been planned in order to achieve the objectives stated in Section 2.1 and respond to the RQs formulated in Section 2.2. For objectives $\mathbf{O}_2$-$\mathbf{O}_5$[1], the following sequence of tasks has been carried out:

1. Literature search and analysis of current approaches related to both the problem domain and the applied technique.
2. Formulation of the SE task as an optimisation problem, including the software metrics required to evaluate their artefacts.

[1]For objective $\mathbf{O}_1$, only the first step was required.

3. Design and implementation of the components of the metaheuristic model, i.e. problem encoding, fitness function, domain-specific search operators and search algorithm.

4. Experimental validation of the approach with input data derived from real software systems, including the study of the influence of their parameters.

5. Integration of the proposed models to be provided as part of a DSS oriented towards software engineers. Development of supporting libraries and tools.

Table 2.1: Objectives, research tasks and publications

<table>
<tr><th></th><th>Literature analysis</th><th>Problem formulation</th><th>Algorithm implementation</th><th>Experimental validation</th><th>Model integration</th></tr>
<tr><td rowspan="2">$\mathbf{O_1}$</td><td>TSE [J4]</td><td colspan="4" rowspan="2"></td></tr>
<tr><td>UnderReview1</td></tr>
<tr><td>$\mathbf{O_2}$</td><td>JISBD'14 [C2]</td><td colspan="3">INFSCI [J1]</td><td></td></tr>
<tr><td rowspan="3">$\mathbf{O_3}$</td><td rowspan="3"></td><td colspan="3">GECCO'14 [C1], MAEB'15 [C3], EMSE [J2]</td><td>GECCO'15 [C4]</td></tr>
<tr><td colspan="3">JISBD'16 [C6], ISDA'16 [C9], CEC'17 [C10]</td><td>UnderReview2</td></tr>
<tr><td>JISBD'16 [C7]</td><td colspan="2">ESwA [J3]</td><td></td></tr>
<tr><td rowspan="3">$\mathbf{O_4}$</td><td rowspan="3"></td><td>JISBD'15 [C5]</td><td colspan="3"></td></tr>
<tr><td colspan="3">INFSCI [J5]</td><td>MAEB'16 [C8]</td></tr>
<tr><td>JISBD'17 [C11]</td><td colspan="2"></td><td>JISBD'18 [C12]</td></tr>
</table>

The results of this Ph.D. Thesis are supported by seven journal articles: the three articles comprising the compendium, two articles as a result of both Spanish and international research collaborations, and two works currently under review, all of them published (or submitted) to reference journals[2] within both AI and SE fields. Furthermore, four and eight conference papers, some of which were nominated for *best paper*, have been published in international and national conferences, respectively. It is worth mentioning that the Thesis proposal was awarded a second prize in the first Doctoral Consortium organised by the Spanish Society of Software Engineering and Software Development Technologies (SISTEDES).

Table 2.1 shows the publications derived from the Ph.D. Thesis, grouped by the objective and research task(s) to which they are attributed. In the table, J stands for references to journal publications, while C represents references to conference

[2]All journals are ranked at the first quartile according to the Journal Citation Reports.

publications whether in national or international forums. Those journal articles comprising the compendium are highlighted in bold typeface, whereas italics is used to represent journal articles currently under revision[3]. Publications are identified with the acronym of the journal or conference, and ordered by publication year.

[3] *UnderReview1* refers to a survey of MaOO in the context of SBSE. *UnderReview2* corresponds to a software library for MaOO.

3

Methodology

"Reserve your right to think, for even to think wrongly
is better than not to think at all."
Hypatia

This chapter details the methodology followed in this Ph.D. Thesis, which mainly concerns the experimental evaluation of the different algorithms proposed. Also, best practices in SE research have been adopted regarding literature search and discussion of validity threats. The following sections briefly describe all these methodological aspects, for which further explanations can be found in the corresponding publications.

3.1. Literature analysis

Along the development of the Ph.D. Thesis, literature searches have been periodically performed to find relevant works regarding both the application domain and the selected techniques. The sources of information comprise both general databases (ACM Library, DBLP, IEEE Xplore, ISI Web of Knowledge, ScienceDirect, SpringerLink and Scopus) and a specialised repository of SBSE publications [208].

In addition, a systematic method for literature review [103, 104] has been followed to analyse the application of advanced techniques, i.e. MaOO, and interactive optimisation in the context of SBSE. The three main phases of a systematic literature review (SLR) are: 1) *planning the review*, which consists in defining a review protocol and classification scheme; 2) *conducting the review*, which includes the selection of primary studies and the data extraction process; and 3) *reporting the review*, which refers to the statistical analysis of the extracted data and the discussion of findings.

3.2. Experimental framework

The methodology followed to achieve objectives $\mathbf{O}_2$-$\mathbf{O}_5$ (see Section 2.1) is based on the usual steps of the *scientific method*, as it is often adopted in experimental computer science [58]:

- *Observation.* Study of the needs of automation in software architecture design.
- *Induction.* Extraction of decision variables and metrics from real-world scenarios in order to define software architecture optimisation problems.
- *Hypothesis.* Formulation of the RQs and design of metaheuristic models to address them.
- *Experimentation.* Evaluation of the performance and utility of the proposed algorithms using real software systems.
- *Antithesis.* Analysis of the obtained results with respect to the RQ.

The rest of this section details the characteristics of the experimental phase: the computing environment to code and run the algorithms, the software systems used as problem instances and the evaluation mechanisms needed to assess the performance of the algorithms. It should be noted that the validation of the interactive model described in Chapter 4 slightly differs from the rest of experiments. In that case, the principal experiment consisted in an empirical study in which software developers were asked to participate in interactive sessions. Details of the experiment preparation and the analysis of participants' actions and feedback can be found in the journal publication associated to that model [J5].

3.2.1. Implementation and execution environments

Software libraries and tools

All algorithms have been coded in Java using *JCLEC* [193], a Java library for EC, and *JCLEC-MO*, an extension for MOO that has been developed as part of this Ph.D. Thesis[1]. *Datapro4j* [165] and *SDMetrics Open Core* [172] are supporting libraries used to process data and extract analysis information from XMI files, respectively. The *R statistical environment* [156] and *STATService* [144] have provided the statistical tests applied for performance assessment.

Hardware platform

For experimentation, machines with 8 cores Intel Core i7 2.67-GHz and 12GB RAM and hosting Linux distributions (*Ubuntu*, *Debian*) have been used. The most extensive experiments have been run on a HPC cluster of 8 compute nodes with Rocks cluster 6.1 x64 Linux distribution, each node comprising two Intel Xeon E5645 CPUs with 6 cores at 2.4 GHz and 24 GB DDR memory.

3.2.2. Problem instances

Table 3.1 shows the characteristics of the software systems used for experimentation. They all are real systems or libraries written in Java, with the exception of *Aqualush*, which is a case study for educational purposes. Relationships are divided into the different types defined by UML, i.e. associations, dependencies, aggregations, compositions and generalisations. The analysis models for these systems were obtained after a thorough manual analysis of the source code and design documentation, when available.

3.2.3. Performance evaluation

In general, the performance of SBSE methods is quantitatively evaluated following the same guidelines than other fields applying metaheuristics. Due to the stochastic nature of this type of algorithms, their performance cannot be inferred from the

[1] A preliminary version was published at GECCO conference [C4], whereas an article describing its complete architecture is currently under revision.

Table 3.1: Software systems used for experimentation

System	**#Classes**	**#Relationships**					**#Interfaces**
		Assoc.	*Depen.*	*Agreg.*	*Comp.*	*Gener.*	
Aqualush	58	69	6	0	0	20	74
Borg	167	44	109	36	38	90	300
Datapro4j	59	3	4	3	2	49	12
ICal4j	190	36	4	3	11	161	70
Java2HTML	53	20	66	15	0	15	170
JSapar	46	7	33	21	9	19	80
JXLS	96	60	10	10	9	45	136
Logisim	253	113	19	46	25	137	248
Marvin	32	5	11	22	5	8	28
NekoHTML	47	6	17	15	18	17	46

outcomes of a single run. For this reason, the common practice within the field is to perform several runs with different random seeds and report average results. A minimum of 30 runs is often suggested [84]. From these runs, different performance measures can be obtained so that performance can be studied from diverse and complementary views. More specifically, best or average fitness values at the end of the evolution are usually reported for single-objective algorithms. For MOEAs, quality indicators have been also adopted by the SBSE community [200]. In this Ph.D. Thesis, hypervolume, spacing and coverage have been chosen (see Table 1.1). Other aspects of great importance in SBSE are the number of solutions returned to the engineer at the end of the process, and the execution time required to find them. Both aspects have been investigated in the experimental studies, since they have implications in the practicality of the proposed methods.

After the collection of these measurements, the relative performance of two or more metaheuristic algorithms can be compared using statistical methods, a type of analysis that is highly recommended in SBSE too [12]. Statistical tests serve to estimate the confidence in the obtained results and prove the existence of significant differences between them [187]. More specifically, a statistical test verifies whether the *null hypothesis* (H_0) – that states that the probability distributions of the samples are equivalent – can be rejected. If so, there is statistical evidence to suggest that significant differences exist. Two types of errors can appear as a consequence of a

wrong estimation. Type I error occurs when H_0 is rejected but it is actually true, whereas Type II error indicates that H_0 is mistakenly accepted [12]. The outcome of a statistical test is the probability of Type I error, known as *p-value*, so lower values are preferred. The highest *p-value* that can be accepted to reject H_0 is called *significant level*, α. Depending on the nature of the test, some of them can also rank algorithms according to their performance or report the magnitude of their differences (effect size), thus making interpretation of results easier. For metaheuristic algorithms, non-parametric tests are frequently applied, as they do not require any assumption about the distribution of the samples, while H_0 is usually formulated to express that two or more algorithms perform equally well with respect to a selected measure, e.g. hypervolume. In this Ph.D. Thesis, the following non-parametric tests have been considered:

- *Wilcoxon rank sum* test [203] for pairwise comparison.
- *Friedman* test [71] with *Holm post hoc* procedure [91] for multiple comparison.
- *Cliff's delta* test [164] for effect size measurement.

3.3. Threats to validity

The analysis of validity threats is a common practice within empirical SE [66] that has been progressively adopted in SBSE studies [16]. When applicable, aspects related to the four types of validity threats have been discussed in the publications derived from this Ph.D. Thesis:

- *Construct validity*, which is focused on the relation between the theory and the observation(s). In SBSE, potential threats arise from the assumptions and simplifications regarding the problem formulation.
- *Internal validity*, which is related to the causal relation between the hypothesis and the results. In SBSE, this type of validity is mainly affected by algorithm parametrisation.
- *Conclusion validity*, which refers to the relationship between the treatment and the outcome. In SBSE, they can appear if good practices when evaluating algorithm performance are not followed (see Section 3.2.3).

- *External validity*, which is concerned with the generalisation of the results. In SBSE, it is related to the selection of representative problem instances.

4

Results

"However beautiful the strategy, you should occasionally look at the results."
Unknown

This chapter presents the proposed models and a summary of the most relevant results obtained from the experiments carried out to validate them. Firstly, the single-objective evolutionary approach to address the discovery of component-based software architectures is described. The extension of this model considering a greater number of software metrics as independent objectives is detailed next. Lastly, the main characteristics of the interactive approach, which allows the integration of human feedback, are explained. The associated publications to each model are listed at the end of the corresponding section, and the full content of the journal publications can be found in Part II of this document.

4.1. Evolutionary discovery of architectures

The recovery of the main functional blocks, and their interactions, from working systems has only been tackled from a re-engineering perspective, mostly analysing code or other low-level artefacts. However, the system structure derived from this process might not be representative, since analysis information is omitted and source

code might not correspond to the conceived architecture. In contrast, the architecture could be better *discovered* from previous analysis information. Under these premises, a first contribution of this Ph.D. Thesis consists in the formulation of such a process as an optimisation problem. For its resolution, a single-objective evolutionary algorithm with customised operators and a ranking-based fitness function has been developed. The most relevant aspects of the proposed approach and a discussion of the experimental results are presented next.

4.1.1. Proposed approach

This section presents the main components of a novel approach, based on EAs, to find the underlying architecture of a software system [J1]. The discovery process requires an initial UML2 class diagram as starting point, containing information of classes and their relationships. The search process is therefore aimed at discovering the elements that constitute a component-based architecture, i.e. components, interfaces and connectors, represented as a UML2 component diagram. To do this, an optimal arrangement of classes and relationships should be determined, so that the specification of the internal structure of components and the identified interfaces reflect the properties of a good design.

As part of the problem formulation, an analysis of design metrics proposed in the literature has been carried out. The purpose of such an analysis is to study whether they are suitable to evaluate architectural solutions automatically generated [C2]. After this analysis, as well as preliminary experiments, the three design metrics shown in Table 4.1 are proposed. These metrics are the basis of a ranking-based fitness function that evaluates the quality of each solution.

The rest of the components of the single-objective EA are described next:

- Candidate component-based architectures, i.e. individuals, are encoded using tree structures. This way, the hierarchical composition of the architecture can be represented and handled.
- The initial population is randomly created. For each individual, the number of components is chosen at random, between a minimum and a maximum configured by the engineer.

Table 4.1: Design metrics to evaluate component-based architectures (I)

Name	Intra-modular Coupling Density (ICD)
Quality	Modularity
Description	Average ratio between the number of interactions inside (CI_c^{in}) and outside (CI_c^{out}) each component, c. The value is weighted with the ratio of classes that belong to that component ($\#classes_c$) from the total number of classes ($\#classes_{total}$).
Objective	Maximisation
Bounds	[0, 1]
Formulation	$ICD_c = \frac{num\_classes_{total} - num\_classes_c}{num\_classes_{total}} \cdot \frac{CI_c^{in}}{CI_c^{in} + CI_c^{out}}$ $ICD = \frac{1}{C} \cdot \sum_{c=1}^{C} ICD_c$
Name	External Relations Penalty (ERP)
Quality	Modularity
Description	Total number of associations (num_as), aggregations (num_ag), compositions (num_co) and generalisations (num_ge) between each pair of components, c and t, that cannot be abstracted as an interface. Each type of relationship receives a weight (w).
Objective	Minimisation
Bounds	Lower bound is 0, upper bound depends on the input class diagram
Formulation	$ERP = \sum_{c=1}^{C} \sum_{t=c+1}^{C} w_{as} \cdot num\_as_{c,t} + w_{ag} \cdot num\_ag_{c,t} +$ $+ w_{co} \cdot num\_co_{c,t} + w_{ge} \cdot num\_ge_{c,t}$
Name	Groups/Components Ratio (GCR)
Quality	Reusability
Description	Ratio between the number of groups of interdependent classes ($num\_cgroups$) and the total number of components, C.
Objective	Minimisation
Bounds	Lower bound is 1, upper bound depends on the input class diagram
Formulation	$GCR = num\_cgroups/C$

- A mutation operator applies up to five different architectural transformations exploiting domain knowledge. The available transformations are: add a component, split a component, remove a component, merge two components and move a class. The engineer can assign a different weight to each one.

- Several selection and replacement strategies are implemented with the aim of analysing the balance between convergence and diversity. These strategies are those frequently used in EC, such as elitism or tournaments based on fitness values. The best combination is determined as part of the experimental study.

This algorithm has also served as a basis for further studies that explore the capabilities of trajectory-based metaheuristics to improve the evolutionary search [C6, C9]. More specifically, two MAs have been designed: 1) *EA(LS)*, which applies local improvements over a subset of offspring right after mutation; and 2) *EA+LS*, which runs HC or SA after the execution of the EA taking a percentage of the best individuals from the final population as input.

4.1.2. Discussion of results

A parameter study has been carried out to analyse the influence of the parameters required by the evolutionary model [J1]. The main conclusions drawn from this study are summarised next. Firstly the analysis shows that the combination of selection and replacement strategies plays an important role in the behaviour of the algorithm. Due to its proper convergence and trade-off between metrics, a selection operator based on binary tournament and an elitist replacement that preserves 10% of the current population are finally selected.

In addition, the best combination of weights for the five procedures constituting the mutation operator is also explored. In this case, the algorithm has been executed with 126 possible combinations. Does this study not only shed light on the influence of the mutation operator on final fitness values, but also on how its internal procedures promote certain types of solutions regarding the number of components. The last parameters under study are the population size and the number of generations, which are analysed together since both are related to the total number of evaluations allocated to the algorithm. The experimental findings suggest that the algorithm shows a better performance when 150 individuals are evolved during at least 20000 evaluations.

Once the best algorithm configuration was found, further experiments have been conducted to analyse additional aspects. From the obtained results, the behaviour of the algorithm with respect to the optimisation of ERP and ICD metrics clearly reflects the usual conflict between coupling and cohesion. The comparisons between problem instances having similar number of classes also suggest that the complexity of the problem cannot be attributed to the number of classes only, but also to the number and sort of relationships between them. Finally, the ability of the algorithm

to produce architectural specifications close to those manually produced can be also highlighted.

As for the two MAs proposed as extension, the experimental outcomes indicate that the way hybridisation is designed has a considerable impact on algorithm performance [C6, C9]. On the one hand, *EA(LS)* works well when low probabilities are assigned to either HC and SA. However, the original algorithm is preferable when there is no particular trend for a metric, as it maintains a better trade-off among all of them. On the other hand, *EA+LS* does not show great effectiveness compared to the pure EA. Although the final population reflects some local improvements, the best solution found during the evolutionary process is not modified.

To conclude, the proposed model and the experiments carried out have served to respond to the RQs formulated for objective $\mathbf{O}_2$ (see Section 2.2). In response to $\mathbf{RQ}_4$, a comprehensible single-objective evolutionary model is provided, since all its elements were designed taking in mind how an architect would proceed in a manual process. In addition, the complete experimental study provides evidence of the effectiveness of the approach, as well as the importance of tuning its parameters ($\mathbf{RQ}_5$).

4.1.3. Associated publications

The principal publication associated to this section is:

A. Ramírez, J.R. Romero, S. Ventura. *"An approach for the evolutionary discovery of software architectures"*. **Information Sciences**, vol. 305, pp. 234-255, 2015.

The following conference publications present the analysis of software metrics and the hybrid variants of the evolutionary approach:

1. A. Ramírez, J.R. Romero, S. Ventura. *"Análisis de la aplicabilidad de medidas software para el diseño semi-automático de arquitecturas"*. Proceedings of the XIX Jornadas en Ingeniería del Software y Bases de Datos (JISBD), pp. 307-320, 2014.

2. A. Ramírez, J.A. Molina, J.R. Romero, and S. Ventura. *"Estudio de mecanismos de hibridación para el descubrimiento evolutivo de arquitecturas"*. Proceedings of the XXI Jornadas en Ingeniería del Software y Bases de Datos (JISBD), pp. 481-494, 2016.

3. A. Ramírez, R. Barbudo, J.R. Romero, S. Ventura. *"Memetic Algorithms for the Automatic Discovery of Software Architectures"*. Proceedings of the 16th International Conference on Intelligent Systems Design and Applications (ISDA), vol. 557 AISC, pp. 437-447, 2016.

4.2. The multi- and many-objective perspectives

The evolutionary model proposed in Section 4.1 represents an important advancement in automated architecture discovery, but has limited application as supporting method since only one solution is returned at the end of the process. The conclusions drawn from the experimentation also indicate the existence of conflicts between some of the design metrics used to compute the fitness function. In fact, additional metrics could be used to guide the process, making necessary to establish new trade-offs among them. All these aspects motivate the exploration of multi-objective approaches, including recent developments on MaOO. The application of these advanced techniques could provide the expert with a variety of architectural solutions to choose among, while generating more insights on the influence of design metrics in the optimisation process. Next section reformulates the optimisation problem in terms of multiple, independent objectives. Then a summary of the experimental studies carried out to analyse the performance of some selected MOEAs and MaOEAs follows.

4.2.1. Proposed approach

This section presents the adaptations made in the evolutionary model to cope with multiple design objectives [J2]. It should be noted that solution encoding and population initialisation remain as described in Section 4.1.1, since the architectural elements comprising the candidate solutions and how they are derived from the input class diagram are not affected by the evaluation phase.

Table 4.2: Design metrics to evaluate component-based architectures (II)

Name	Instability (INS)
Quality	Modularity
Description	Average component instability, which is obtained as a ratio between afferent (AC_c) and efferent coupling (EC_c). These terms represent the number of components requiring services from c and the number of components providing services to c, respectively.
Objective	Minimisation
Bounds	[0, 1]
Formulation	$INS_c = \frac{EC_c}{EC_c + AC_c} \quad INS = \frac{1}{C} \cdot \sum_{i=1}^{C} INS_c$
Name	Abstractness (ABS)
Quality	Reusability
Description	Average component abstractness, which is calculated as the ratio between abstract classes and total number of classes inside the component.
Objective	Maximisation
Bounds	[0, 1]
Formulation	$ABS_c = \frac{\#abstract\_classes_c}{\#classes_c} \quad ABS = \frac{1}{C} \cdot \sum_{i=1}^{C} ABS_c$
Name	Encapsulation (ENC)
Quality	Modularity
Description	Average component encapsulation, which is computed as the ratio between the number of classes that do not participate in external interactions and the total number of classes inside the component.
Objective	Maximisation
Bounds	[0, 1]
Formulation	$ENC_c = \frac{\#inner\_classes_c}{\#classes_c} \quad ENC = \frac{1}{C} \cdot \sum_{i=1}^{C} ENC_c$

Focusing on solution evaluation, up to nine design metrics have been defined as potential objectives to be simultaneously optimised. Three of them correspond to the metrics already used in the single-objective approach (see Table 4.1). Analogously, other six design metrics measuring aspects beyond cohesion and coupling have been considered. Tables 4.2 and 4.3 show the definition and properties of these new metrics. From the metrics above, all possible combinations of two, four, six, eight and nine metrics – 256 combinations in total – have been defined as the objectives to be optimised by the state-of-the-art algorithms presented in Section 1.2.3.

In addition, the two hybrid models proposed in Section 4.1.1 have been adapted to work with a formulation with four objectives (ICD, ERP, CS and CB). This

Table 4.3: Design metrics to evaluate component-based architectures (III)

Name	Critical Size (CS)
Quality	Modularity
Description	Number of critical components (CC), i.e. those whose size exceeds a threshold, λ. Size is computed as the percentage of classes allocated in c with respect to the total number of classes.
Objective	Minimisation
Bounds	[0, C]
Formulation	$CC_c^{size} = 1\ if\ size(c) > \lambda,\ 0\ otherwise$ $CS = \sum_{i=1}^{C} CC_c^{size}$
Name	Critical Link (CL)
Quality	Modularity
Description	Number of critical components with respect to their interactions, i.e. their number of provided interfaces exceeds a threshold, λ.
Objective	Minimisation
Bounds	[0, C]
Formulation	$CC_c^{link} = 1\ if\ \#provided\_interfaces_c > \lambda,\ 0\ otherwise$ $CS = \sum_{i=1}^{C} CC_c^{link}$
Name	Component Balance (CB)
Quality	Analisability
Description	Measure combining the deviation from an optimal number of components in the architecture (system breakdown, SB) and the dispersion in their size (component size uniformity, CSU). For SB, γ, ω and μ represent the minimum, maximum and optimal number of components, respectively.
Objective	Maximisation
Bounds	[0, 1]
Formulation	$SB = \frac{C-\gamma}{\mu-\gamma}\ \ if\ \gamma \leq C < \mu\ ,\ \ 1 - \frac{C-\mu}{\omega-\mu}\ \ if\ \mu \leq C \leq \omega$ $CSU = 1 - Gini(\#classes_c\ \forall c \in [1, C])\quad CB = SB \cdot CSU$

particular combination provides an appropriate trade-off between cohesion, coupling and size, as will be discussed in Section 4.2.2. Another adaptation concerns the definition of a comparison criterion to decide whether a neighbour, x', is better than the current solution, x, since multiple objectives exist. Four different mechanisms are proposed for comparative purposes: Pareto dominance, meaning that x' is accepted if it dominates x; weights, in which a weighted sum of objective values is computed; best objective, meaning that only the best objective in the initial solution is used for comparison; and worst objective, which adopts the same approach but considering

the worst objective. The selection and replacement mechanisms of NSGA-II are applied during evolution. Apart from HC and SA, TS has been implemented in combination with the four acceptance criteria.

4.2.2. Discussion of results

Extensive experiments were conducted to study the scalability of eight MOEAs and MaOEAs, as well as the influence of the selected metrics in the structure and quality of the solutions [J2]. The most relevant findings obtained from this comparative study are compiled next.

A first analysis in terms of quality indicators yields interesting results regarding algorithm scalability. For two or four objectives, NSGA-II provides the best results in terms of hypervolume, whereas SPEA2 obtains the best spacing values. However, many-objective algorithms like ϵ-MOEA and HypE demonstrate their superiority to converge to the PF with the increase in the number of objectives. The good scalability of NSGA-II is remarkable, being able to ensure a better trade-off between both indicators than specialised many-objective approaches.

A closer look at these results from the point of view of the metrics being optimised allows making further analyses. When a small number of objectives is set, the specific combination of metrics can result in a decrease in problem complexity, since objectives are not strongly opposed. The particular choice of metrics becomes less relevant as the number of objectives is increased. In this scenario, trade-offs among all metrics are difficult to achieve, some metrics being highly optimised to the detriment of others.

The suitability of the multi-objective approach as supporting method for decision-making have been examined from diverse perspectives. Firstly, the number of solutions returned to the engineer depends on both the number of objectives and the selected algorithm. Another relevant factor is execution time, since some many-objective algorithms are computationally expensive.

Finally, it should be noted that the specific combination of metrics inevitably impact the type of architectural solutions found. A combination of cohesion, coupling and size, such as using ICD, ERP, CS and CB as objectives, can work well in general. In fact, NSGA-II guided by this particular combination was able to produce solutions

close or even equal to the known architecture of a system, i.e. designed by a real architect.

The experiments for the MAs have been analysed with respect to both evolutionary performance and number of final solutions [C10]. On the one hand, the execution of trajectory-based methods either during of after the evolution does not decrease efficiency of the evolutionary search according to the coverage indicator. Regarding diversity of solutions, spacing values indicate that the hybrid model could lead to substantial improvements, though the magnitude of the gain depends on the specific parametrisation. On the other hand, trajectory-based methods can help in the generation of new non-dominated solutions, specially if *EA(LS)* is applied. For both MAs, the use of weights to compare the solutions seems to be more effective for this purpose.

Coming back to the RQs formulated for objective O_3 (see Section 2.2), it can be concluded that the discovery process can be guided by many diverse design criteria, whose selection can have a strong impact on the optimisation process ($\mathbf{RQ}_6$). In this sense, MOEAs or even MaOEAs are required to cope with the increasing complexity of the problem ($\mathbf{RQ}_7$). However, the choice of a particular algorithm should not be based on the number of metrics only, but also on additional aspects like execution time and expected number of solutions. Focusing on the effect of LS, its impact can be more effective than in the single-objective formulation, though additional parameters need to be properly configured ($\mathbf{RQ}_8$).

4.2.3. Associated publications

The principal publication associated to this section is:

A. Ramírez, J.R. Romero, S. Ventura. *"A comparative study of many-objective evolutionary algorithms for the discovery of software architectures"*. **Empirical Software Engineering**, vol. 21, no. 6, pp. 2546-2600, 2016.

The following conference publications present preliminary results on the performance of MOEAs and MaOEAs, a software framework to support their implementation, and the adaptation of MAs to the multi-objective problem formulation:

1. A. Ramírez, J.R. Romero, S.Ventura. *"On the Performance of Multiple Objective Evolutionary Algorithms for Software Architecture Discovery"*. Proceedings of the 16th Genetic and Evolutionary Computation Conference (GECCO), pp. 1287-1294, 2014. *Best paper of the SBSE track and nominated to best paper of GECCO'14.*

2. A. Ramírez, J.R. Romero, S.Ventura. *"Estudio preliminar del rendimiento de familias de algoritmos multiobjetivo en diseño arquitectónico"*. Proceedings of the X Congreso Español sobre Metaheurísticas, Algoritmos Evolutivos y Bioinspirados (MAEB), pp. 173-180, 2015.

3. A. Ramírez, J.R. Romero, S.Ventura. *"An Extensible JCLEC-based Solution for the Implementation of Multi-Objective Evolutionary Algorithms"*. Proceedings of the Companion Publication of 17th Genetic and Evolutionary Computation Conference (GECCO Companion), pp. 1085-1092, 2015.

4. A. Ramírez, J.R. Romero, S.Ventura. *"On the effect of local search in the multi-objective evolutionary discovery of software architectures"*. Proceedings of the IEEE Congress on Evolutionary Computation (CEC), pp. 2038-2045, 2017.

4.3. The human-in-the-loop approach

The adaptation of the initial evolutionary approach to a multi-objective space has allowed the introduction of more decision factors, as well as the possibility of returning a set of solutions at the end of the process. Both aspects enrich the decision-making scenario, allowing the software engineer to adapt the discovery process to his/her needs and choose among a set of alternatives, respectively. However, he/she is not yet involved in the optimisation phase. Therefore, this section is focused on how human expertise can be incorporated in the discovery process and the experiments undertaken to validate the approach, including an empirical study with software developers.

4.3.1. Proposed approach

The IEC model is comprised of a customised MOEA, designed according to previous findings and new requirements, and an interaction module that coordinates the communication between the algorithm and the software engineer [J5]. On the one hand, the proposed MOEA follows the evolution scheme of ϵ-MOEA, whose archive has been adapted to store only a small number of representative solutions. The archive update mechanism, inspired by a general-purpose interactive EA named iTDEA [105], keeps a portion of diverse non-dominated solutions, as well as those solutions of interest to the engineer. On the other hand, the interaction module is characterised by the following aspects:

- Interactions are scheduled at regular intervals, though the number of interactions is configurable.
- The number of solutions to be shown in each interaction is also a parameter. Solutions are selected from the current population using a clustering method.
- In each interaction, the software engineer is asked to choose one *architectural preference*, among those described in Table 4.4, in order to reward or penalise some aspect of the solution.
- The software engineer can also perform additional actions: *freeze* one component, so that its classes cannot be moved; add the solution to the archive; remove the solution from the current population; and stop the search.

The fitness function combines qualitative and quantitative information, using a weighted sum of two terms. The former determines the extent to which a candidate solution satisfies the preferences established by the software engineer. The second term makes use of the *maximin* function [15] to evaluate the dominance of the solutions according to the configured software metrics.

4.3.2. Discussion of results

To validate the proposed interactive approach, two experiments have been carried out [J5]. The first experiment is aimed at studying the evolutionary performance,

Table 4.4: Design preferences for the interactive discovery of architectures

Preference	**Description**
Best component	The engineer selects the best component according to its internal structure.
Worst component	The engineer indicates the worst component.
Best provided interface	The engineer identifies the best interface according to its operations.
Worst provided interface	The engineer locates the worst interface.
Number of components	The engineer expresses the number of components for the architecture he/she considers appropriate.
Metric in range	The engineer establishes the desired range for a metric.
Aspiration levels	The engineer determines weights and target values for all design metrics.

so interaction is omitted. Firstly, the influence of the parameter that controls the archive size is analysed. The main conclusion obtained here is that an intermediate value is preferred since it provides a good performance, while returning a number of solutions that is affordable for manual inspection.

Secondly, the algorithm is compared against NSGA-II, which is a representative MOEA that has proven to be highly competitive. In this sense, the experimental results indicates that NSGA-II is slightly superior with respect to hypervolume, but obtains lower spacing values. It should be noted that these results are somehow conditioned by the fact that NSGA-II returns a larger number of non-dominated solutions. This suggests that the proposed algorithm is more effective as supporting method for decision-making, since it provides equivalent performance even if the number of solutions has to be limited.

Interactive sessions with participants of different expertise in software development have also been planned in order to assess the effectiveness and usefulness of the approach. In these sessions, participants were asked to interact with a tool running the algorithm to solve a case study. Some interesting facts can be extracted from these sessions. Firstly, the most frequently applied architectural preferences were selecting the best/worst component or setting the appropriate number of components. They were also viewed as the most intuitive and useful options. In occasions, the participants decided not to select any particular preference as a way to deal with uncertainty or because they wanted to observe how the algorithm behaves.

Comparing the performance of the algorithm with and without interaction, both variants behave quite similar in terms of quality indicators. This suggests that the combination of quantitative and qualitative criteria is effective to guide the process while adapting the search to the engineer's preferences. Focusing on the design metrics under evaluation (see Table 4.1), the actions performed by the participants were indirectly oriented towards improving cohesion. Furthermore, the influence of human interaction was also reflected in the number of components comprising final solutions. When the participants indicated the preferred number of components, the algorithm rapidly converged to solutions satisfying this preference.

To conclude, the RQs referring to the interactive approach have been fully addressed. On the one hand, the proposed model not only allows the engineer to evaluate solutions, but also perform actions with a direct impact on the search algorithm ($\mathbf{RQ}_9$). On the other hand, the empirical study has demonstrated how interaction positively affects the optimisation process ($\mathbf{RQ}_{10}$).

4.3.3. Associated publications

The principal publication associated to this section is:

A. Ramírez, J.R. Romero, and S. Ventura. *"Interactive multi-objective optimization of software architectures"*. **Information Sciences**, vol. 463-464, pp. 92-109, 2018.

In addition, a SLR covering the whole field of SBSE with respect to the application of interactive methods has been published in *IEEE Transactions of Software Engineering* in collaboration with Dr. Christopher L. Simons (University of the West of England, Bristol, UK). Other conference publications refer to preliminary studies of the suitability of interactive approaches to the discovery problem, as well as supporting libraries and tools for experimentation:

1. A. Ramírez, J.R. Romero, C. Simons. *"A Systematic Review of Interaction in Search-Based Software Engineering"*. IEEE Transactions on Software Engineering, pp. 1-22. 2018. *In press.*

2. A. Ramírez, J.R. Romero, S.Ventura. *"Interactividad en el descubrimiento evolutivo de arquitecturas"*. Proceedings of the XX Jornadas en Ingeniería del Software y Bases de Datos (JISBD), 2015.

3. A. Ramírez, R. Barbudo, J.R. Romero, S. Ventura. *"Herramienta basada en computación evolutiva interactiva para arquitectos software"*. Proceedings of the XI Congreso Español sobre Metaheurísticas, Algoritmos Evolutivos y Bioinspirados (MAEB), pp. 387-396, 2016.

4. A. Ramírez, J.R. Romero, S. Ventura. *"Búsqueda coevolutiva interactiva aplicada al diseño de software"*. Proceedings of the XXII Jornadas en Ingeniería del Software y Bases de Datos (JISBD), 2017.

5. A. Ramírez, J.R. Romero, S. Ventura. *"API para el desarrollo de algoritmos interactivos en ingeniería del software basada en búsqueda"*. XXIII Jornadas en Ingeniería del Software y Bases de Datos (JISBD), 2018.

5

Conclusions and future work

"The mind that opens to a new idea never returns to its original size."
Albert Einstein

Throughout the development of this Ph.D. Thesis, a number of contributions to SBSD have been made. These contributions can be viewed from diverse perspectives, since both SE optimisation problems and search algorithms have been proposed. In this sense, Section 5.1 compiles the conclusions that can be drawn from the different research works presented in this Ph.D. Thesis. The novelty of the problem addressed, together with the application of incipient techniques within the field of SBSE, suggest that this Ph.D. Thesis could serve as a basis for further investigation. Therefore, Section 5.2 points out future lines of research, which mainly concern the extension of the problem formulation, the exploration of additional techniques, and the application of the proposed models to other design tasks.

5.1. Concluding remarks

This Ph.D. Thesis has explored the use of metaheuristic techniques as effective and efficient methods for decision support in the context of early software analysis. Framed within the field of SBSD, a new SE optimisation problem, named

discovery of component-based software architectures, has been defined and several search techniques have been applied to its resolution. More specifically, the following contributions can be highlighted:

Definition of software architecture discovery as an optimisation problem. With the aim of providing semi-automatic support to engineers during software analysis, the identification of architectural artefacts from previous analysis models has been formulated as an optimisation problem. Standards for software modelling and evaluation have been followed to create the conceptual framework. Thus, the artefacts and software metrics required to address the discovery process are representative of the elements a software engineer would adopt in a manual process.

Design of a comprehensible evolutionary model to discover architectures. Built upon the aforementioned conceptual framework, a single-objective EA has been proposed to automatically search the underlying architecture of real software systems. A human-centred evolutionary model has been conceived, whose mutation operator imitates the architectural transformations usually performed by engineers. The ranking-based fitness function allows the evaluation of three design metrics related to coupling and cohesion criteria. A thorough analysis of existing parameters also provides useful guidelines to adapt some of the elements to the characteristics of the input system. The proposed model represents a novel approach to the field of software architecture optimisation, in which component-based architectures have usually been optimised at lower abstraction levels.

Analysis of the influence of software metrics in a many-objective space. The evolutionary model has been extended to cope with a larger number of design metrics, thus allowing to include additional quality criteria in the decision process. The performance of state-of-the-art MOEAs and MaOEAs has been assessed in scenarios with different number and combination of metrics. The outcomes reveal the importance of choosing an appropriate subset of metrics to guide the process. Furthermore, additional factors, such as execution time and number of returned solutions, might limit the capabilities of these algorithms as effective supporting methods.

Furthermore, many-objective algorithms have been explored in the context of another design problem too. For the QoSWSC problem, the experimental results reveal that MaOEAs are effective methods when a large number of QoS properties has to be optimised. These works constitute two of the first studies adopting MaOO techniques in SBSE.

Exploration of hybrid approaches to improve the performance of EAs. Both single- and multi-objective evolutionary models have been adapted to include local improvements by means of trajectory-based metaheuristics. Two different MAs have been proposed, one that incorporates LS as an additional genetic operator and another that executes it after the evolution. The experiments carried out to analyse the influence of the parameters suggest that the design of MAs is not straightforward. The results indicate that the moment in which LS is conducted, how often it is applied and the chosen technique can greatly affect the performance of the algorithm.

Development of a general interactive model to integrate human expertise. As architectural analysis demands creativity and experience, a human-in-the-loop approach has been proposed so that the engineer can be fully involved in the optimisation process. The analysis of the interaction requirements has led to the development of an evaluation mechanism that effectively combines quantitative and qualitative design criteria. The innovative idea of expressing the engineer's opinion by means of architectural preferences could be easily adapted to other domains within SBSD. Built upon the knowledge gathered from previous studies, a customised MOEA is proposed. An empirical study with engineers demonstrates how the evolutionary search can be adapted to their needs.

Furthermore, the SLR of human-in-the-loop approaches for SBSE offers an up-to-date analysis of the current state of the field. This is an emerging area with great potential, for which the term *interactive search-based software engineering* (iSBSE) has been coined [J4]. This work also provides useful guidelines to SBSE researchers with the aim of promoting a broader adoption of interactive approaches.

5.2. Future lines of research

The contributions of this Ph.D. Thesis lay the foundations to continue exploring the potential of metaheuristics for software architecture optimisation. In addition, most of the proposed models are highly adaptable to other domains, which clearly open up new possibilities regarding their applicability to other areas of SBSD. Finally, future work should also be oriented towards the integration of these search models into intuitive tools and their validation in industrial contexts. Both aspects are clearly necessary to reduce the current gap between SBSE research and software industry. In the following sections, each of these topics is discussed in more detail.

Design constraints in the discovery process

The current formulation of the automatic discovery of software architectures presents some limitations regarding the input analysis model and the candidate architecture returned as output. Therefore, it could be extended to support additional design elements that frequently appear in real software design scenarios.

Firstly, the input model could present legacy systems that should be properly integrated in the architectural solution. This type of system is often left aside in architecture identification methods [24], though some automatic methods partially consider them [88]. In the discovery process, the specification of the legacy system could be directly modelled in the input class diagram, also expressing which classes depend on it. Then, the EA could use this information to generate the candidate architectures, establishing proper interfaces to communicate the resulting components with the legacy system.

Another possible extension in the current problem formulation refers to the adoption of architectural styles or patterns, as initially explored in some studies [20, 204]. An potential application domain would be the migration to a different architectural styles, e.g. from a client/server architecture to a solution based on microservices. In this case, additional constraints could be defined to express the extend to which a solution complies with a particular style. It would also require the modification of mutation procedures to guarantee that new solutions preserve the style. Depending on the characteristics of the selected styles, adaptations in the encoding to represent them might be needed.

Two additional aspects could be included in the problem formulation. On the one hand, the creation of subcomponents could be an interesting option to deal with large software systems. This would require additional levels of decomposition in the tree structure. On the other hand, the discovery process could be completed with mechanisms to identify the presence of architectural bad-smells. In this sense, avoiding what it is not desired could be a first step towards obtaining an acceptable design.

Evaluation and semantic aspects of software architectures

Software metrics are essential for the evaluation phase of any SBSE approach. However, metrics need to present some properties to be eligible as effective fitness functions, such as approximate continuity and low computational complexity [80]. The nature of the SE task clearly influences both the availability of metrics and the consensus on their practical value, software design being a particularly challenging domain in this regard. More studies focused on the definition of design metrics to measure quality aspects of software architectures are required, as a pre-requisite to their adoption as evaluation criteria in SBSE.

In the same line, it is hard to imagine how the quality of a software design can be measured only in terms of structural properties. The semantic behind the design concepts modelled by software engineers could provide relevant information to the discovery process too. There are some initial attempts to include semantic aspects in automated SE, but mainly referred to code artefacts in the context of modularisation [19, 43], refactoring [141] and architecture recovery [39]. It would be interesting to analyse how semantic information could be inferred at a higher level of abstraction, as well as to study how it might be integrated into an architecture optimisation method.

Advanced interactive models for architecture optimisation

The great potential of interactive optimisation also opens up new possibilities regarding the search models. In current interactive SBSE methods, the engineer participates in specific moments of the evolution, meaning that the overall process is still controlled by the search algorithm. User experience would greatly benefit from a more flexible interaction mechanism, in which the engineer can influence other

aspects of the search beyond evaluation. Letting the user play the central role in the optimisation process is the idea behind hyperinteractive evolutionary computation [33], which remains yet unexplored in SBSE. This advanced IEC model gives the user the opportunity to select when and how genetic operators will be applied.

In addition, the proposed interactive model could be a basis to design new ways of collaboration between the algorithm and the engineer. Cooperative coevolution is an interesting paradigm that could be combined with interactive optimisation in this regard. Coevolutionary systems have proven to be effective for solving difficult optimisation problems [62]. Cooperative coevolution is a variant that follows a *divide-and-conquer* strategy, meaning that each population solves a part of the problem. This idea perfectly fits with the precepts of CBSE, since the architecture is comprised of a set of components that *cooperate* to create a global architecture. Furthermore, humans also adopt a decomposition strategy when facing complex tasks, as it is often the case of architectural analysis. Therefore, an interactive cooperative coevolutionary algorithm seems to be a promising approach to tackle software architecture optimisation problems.

Finally, one important aspect in architectural analysis is that decisions rarely depend on a architect but a design team [162]. Consequently, the interaction mechanism could be extended to support the intervention of more than one person. In such a case, the algorithm would need to find those solutions presenting their common preferences or establish trade-offs among different opinions. Methods from group decision-making could be investigated with this aim. Also, the interactive model could be enhanced with machine learning techniques, which has been proposed in other domains within SBSE as a way to reduce the fatigue caused by constant evaluation [5, 10]. However, these techniques could also be used to learn from the interactive experience, allowing the algorithm to infer user's preferences and recommend solutions of his/her interest.

Resolution of new optimisation problems within SBSD

CBSE is only one of the paradigms conceived for architectural analysis, but many others exist [18]. With the rise of distributed computing, service-oriented architecture (SOA), and particularly its implementation via web services, represents an essential paradigm in the development of modern software systems [143]. The methodology followed here to obtain comprehensible search models for the optimisation of component-based software architectures could be applied to find SOA solutions. In this context, the search process should consider design principles like autonomy, loose coupling an abstraction, being guided by new quality attributes specially relevant to SOA, such as interoperability, reliability, security and scalability, among others [136].

Similarly, the increasing interest in the adoption of cloud-based solutions by leading companies also pose challenges to software development. Decision-making in cloud environments is essential, since engineers have to identify a suitable infrastructure, configure it according to user needs and optimise available resources. Therefore, cloud engineering is a natural scenario for the application of MCDM methods [11] and SBSE techniques [82]. In this sense, the proposed models based on many-objective and interactive optimisation could inspire new methods adapted to the challenges of cloud computing. Examples of potential applications might be the migration of software architectures to the cloud ecosystem, as well as the adoption of cloud design patterns and other best practices during the design process.

Software tools for industrial acceptance

Although research in SBSE has gained maturity, the adoption of its optimisation methods in industrial environments is still limited. Among other practices, the validation of SBSE proposals with case studies from industry and the development of aiding tools are needed to reduce the existing barrier. In fact, several authors, based on their industrial experiences, have reported on the need of reliable and usable tools [70, 122]. In this sense, the proposed models consider SE standards and notations with the purpose of making tool integration easier, and a web-based DSS is currently under development. Future work should be oriented towards the application of the proposed models to industrial case studies and the validation of

the tool with practitioners. In addition, a closer collaboration between academia and industry would allow the identification of new stimulating challenges for SBSE.

Bibliography

[1] M. Abdellatief, A. B. M. Sultan, A. A. A. Ghani, and M. A. Jabar. A mapping study to investigate component-based software system metrics. *Journal of Systems and Software*, 86(3):587–603, 2013.

[2] G. Abowd, R. Allen, and D. Garlan. Using Style to Understand Descriptions of Software Architecture. In *Proceedings of the 1st ACM SIGSOFT Symposium on Foundations of Software Engineering (SIGSOFT)*, pages 9–20, 1993.

[3] S. F. Adra and P. J. Fleming. Diversity Management in Evolutionary Many-Objective Optimization. *IEEE Transactions on Evolutionary Computation*, 15(2):183–195, 2011.

[4] A. Aleti, B. Buhnova, L. Grunske, A. Koziolek, and I. Meedeniya. Software Architecture Optimization Methods: A Systematic Literature Review. *IEEE Transactions on Software Engineering*, 39(5):658–683, 2013.

[5] B. Amal, M. Kessentini, S. Bechikh, and J. Dea. On the Use of Machine Learning and Search-Based Software Engineering for Ill-Defined Fitness Function : A Case Study on Software Refactoring. In *Proceedings of the 6th International Symposium on Search Based Software Engineering (SSBSE)*, pages 31–45, 2014.

[6] D. Ameller, C. P. Ayala, J. Cabot, and X. Franch. Non-functional Requirements in Architectural Decision Making. *IEEE Software*, 30(2):61–67, 2013.

[7] D. Ameller and X. Franch. Assisting software architects in architectural decision-making using Quark. *CLEI Electronic Journal*, 17(3), 2014.

[8] G. Ammons, R. Bodík, and J. R. Larus. Mining Specifications. In *Proceedings of the 29th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL)*, pages 4–16, 2002.

[9] G. Arango, I. Baxter, and P. Freeman. A Framework for Incremental Progress in the Application of Artificial Intelligence to Software Engineering. In *Artificial Intelligence and Software Engineering*, chapter 20, pages 425–438. Ablex Publishing Corporation, 1991.

[10] A. A. Araújo, M. Paixao, I. Yeltsin, A. Dantas, and J. Souza. An architecture based on interactive optimization and machine learning applied to the next release problem. *Automated Software Engineering*, 24(3):623–671, 2017.

[11] J. Araujo, P. Maciel, E. Andrade, G. Callou, V. Alves, and P. Cunha. Decision making in cloud environments: an approach based on multiple-criteria decision analysis and stochastic models. *Journal of Cloud Computing*, 7(1):7, 2018.

[12] A. Arcuri and L. Briand. A Hitchhiker's guide to statistical tests for assessing randomized algorithms in software engineering. *Software Testing, Verification and Reliability*, 24(8):591–592, 2014.

[13] J. Bader and E. Zitzler. HypE: an algorithm for fast hypervolume-based many-objective optimization. *Evolutionary computation*, 19(1):45–76, 2011.

[14] P. Baker, M. Harman, K. Steinhofel, and A. Skaliotis. Search Based Approaches to Component Selection and Prioritization for the Next Release Problem. In *Proceedings of the 22nd IEEE International Conference on Software Maintenance (ICSM)*, pages 176–185, 2006.

[15] R. Balling and S. Wilson. The maximin fitness function for multi-objective evolutionary computation: application to city planning. In *Proceedings of the 3rd Genetic and Evolutionary Computation Conference (GECCO)*, pages 1079–1084, 2001.

[16] M. Barros and A. Dias Neto. Threats to Validity in Search-based Software Engineering Empirical Studies. Technical Report 0006/2011, Universidade Federal do Estado do Rio de Janeiro, 2011.

[17] J. Barthélemy, R. Bisdorff, and G. Coppin. Human centered processes and decision support systems. *European Journal of Operational Research*, 136(2):233–252, 2002.

[18] L. Bass, P. Clements, and R. Kazman. *Software Architecture in Practice*. Addison-Wesley Professional, 3rd edition, 2012.

[19] G. Bavota, A. De Lucia, A. Marcus, and R. Oliveto. Using structural and semantic measures to improve software modularization. *Empirical Software Engineering*, 18(5):901–932, 2013.

[20] A. B. Belle, G. El Boussaidi, C. Desrosiers, S. Kpodjedo, and H. Mili. The Layered Architecture Recovery as a Quadratic Assignment Problem. In *European Conference on Software Architecture*, pages 339–354, 2015.

[21] P. Berander and A. Andrews. Requirements Prioritization. In *Engineering and Managing Software Requirements*, chapter 4, pages 69–94. Springer Berlin Heidelberg, 2005.

[22] M. F. Bertoa, J. M. Troya, and A. Vallecillo. Measuring the usability of software components. *Journal of Systems and Software*, 79(3):427–439, 2006.

[23] D. Birkmeier. On Component Identification Approaches – Classification, State of the Art, and Comparison. In *Proceedings of the 12th International Symposium on Component-Based Software Engineering (CBSE)*, volume 5582 LNCS, pages 1–18, 2009.

[24] D. Q. Birkmeier and S. Overhage. A method to support a reflective derivation of business components from conceptual models. *Information Systems and e-Business Management*, 11(3):403–435, 2013.

[25] C. Blum, J. Puchinger, G. R. Raidl, and A. Roli. Hybrid metaheuristics in combinatorial optimization: A survey. *Applied Soft Computing*, 11(6):1–10, 2012.

[26] C. Blum and A. Roli. Metaheuristics in Combinatorial Optimization: Overview and Conceptual Comparison. *ACM Computing Surveys*, 35(3):268–308, 2003.

[27] I. Boussaïd, J. Lepagnot, and P. Siarry. A survey on optimization metaheuristics. *Information Sciences*, 237:82–117, 2013.

[28] I. Boussaïd, P. Siarry, and M. Ahmed-Nacer. A survey on search-based model-driven engineering. *Automated Software Engineering*, 24(2):233–294, 2017.

[29] E. Bouwers, J. Correia, A. van Deursen, and J. Visser. Quantifying the analyzability of software architectures. In *Proceedings of the 9th Working IEEE/IFIP Conference on Software Architecture (WICSA)*, pages 83–92, 2011.

[30] M. Bowman, L. C. Briand, and Y. Labiche. Solving the class responsibility assignment problem in object-oriented analysis with multi-objective genetic algorithms. *IEEE Transactions on Software Engineering*, 36(6):817–837, 2010.

[31] J. Branke, K. Deb, K. Miettinen, and R. Slowiński, editors. *Multiobjective Optimization: Interactive and Evolutionary Approaches*. Springer-Verlag Berlin Heidelberg, 2008.

[32] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, and M. Stal. *Patern-Oriented Software Architecture: A System of Patterns*, volume 1 of *Software design patterns*. Wiley, 1996.

[33] B. J. Bush and H. Sayama. Hyperinteractive evolutionary computation. *IEEE Transactions on Evolutionary Computation*, 15(3):424–433, 2011.

[34] G. Canfora, M. D. Penta, R. Esposito, and M. L. Villani. An approach for QoS-aware service composition based on genetic algorithms. In *Proceedings of the 7th Genetic and Evolutionary Computation Conference (GECCO)*, pages 1069–1075, 2005.

[35] S. Chand and M. Wagner. Evolutionary many-objective optimization: A quick-start guide. *Surveys in Operations Research and Management Science*, 20(2):35–42, 2015.

[36] S. Chardigny and A. Seriai. Quality-driven extraction of a component-based architecture from an object-oriented system. In *Proceedings of the 12th European Conference on Software Maintenance and Reengineering (CSMR)*, pages 269–273, 2008.

[37] S. Chardigny and A. Seriai. Software Architecture Recovery Process Based on Object-Oriented Source Code and Documentation. In *Proceedings of the 4th European Conference on Software Architecture (ECSA)*, pages 409–416, 2010.

[38] S. Chardigny, A. Seriai, M. Oussalah, and D. Tamzait. Search-Based Extraction of Component-Based Architecture from Object-Oriented Systems. In *Proceedings of the 2nd European Conference on Software Architecture (ECSA)*, volume 5292, pages 322–325. Springer Berlin Heidelberg, 2008.

[39] S. Chardigny, A. Seriai, M. Oussalah, and D. Tamzalit. Extraction of Component-Based Architecture from Object-Oriented Systems. In *Seventh Working IEEE/IFIP Conference on Software Architecture (WICSA 2008)*, pages 285–288, 2008.

[40] J. Clarke, J. J. Dolado, M. Harman, R. M. Hierons, B. Jones, M. Lumkin, B. S. Mitchell, and S. Mancoridis. Reformulating software engineering as a search problem. *IEEE Proceedings Software*, 150(3):161–175, 2003.

[41] C. A. Coello Coello. Evolutionary Multiobjective Optimization: Current and Future Challenges. In *Advances in Soft Computing*, pages 243–256, 2003.

[42] C. A. Coello Coello, G. Lamont, and D. van Veldhuizen. *Evolutionary Algorithms for Solving Multi-Objective Problems*. Springer US, 2nd edition, 2007.

[43] A. Corazza, S. Di Martino, V. Maggio, and G. Scanniello. Weighing lexical information for software clustering in the context of architecture recovery. *Empirical Software Engineering*, 21(1):72–103, 2016.

[44] V. Cortellessa, F. Marinelli, and P. Potena. An optimization framework for "build-or-buy" decisions in software architecture. *Computers and Operations Research*, 35(10):3090–3106, 2008.

[45] M. Cremene, M. Suciu, D. Pallez, and D. Dumitrescu. Comparative analysis of multi-objective evolutionary algorithms for QoS-aware web service composition. *Applied Soft Computing*, 39:124–139, 2016.

[46] J. W. Creswell. *Research Design: Qualitative, Quantitative, and Mixed Methods Approaches*. SAGE Publications, 3rd edition, 2003.

[47] J. F. Cui and H. S. Chae. Applying agglomerative hierarchical clustering algorithms to component identification for legacy systems. *Information and Software Technology*, 53(6):601–614, 2011.

[48] A. S. da Silva, H. Ma, and M. Zhang. Genetic programming for QoS-aware web service composition and selection. *Soft Computing*, 20(10):1–17, 2016.

[49] S. Dasanayake, J. Markkula, S. Aaramaa, and M. Oivo. Software Architecture Decision-Making Practices and Challenges: An Industrial Case Study. In *Proceedings of the 24th Australasian Software Engineering Conference (ASWEC)*, pages 88–97, 2015.

[50] A. De Campos Jr., A. T. R. Pozo, S. R. Vergilio, and T. Savegnago. Many-Objective Evolutionary Algorithms in the Composition of Web Services. In *Proceedings of the 11th Brazilian Symposium on Neural Networks (SBRN)*, pages 152–157, 2010.

[51] K. Deb. *Multi-Objective Optimization using Evolutionary Algorithms*. Wiley Interscience Series in Systems and Optimization. Wiley, 2001.

[52] K. Deb and H. Jain. An Evolutionary Many-Objective Optimization Algorithm Using Reference-Point-Based Nondominated Sorting Approach, Part I: Solving Problems With Box Constraints. *IEEE Transactions on Evolutionary Computation*, 18(4):577–601, 2014.

[53] K. Deb, M. Mohan, and S. Mishra. Towards a Quick Computation of Well-Spread Pareto-Optimal Solutions. In *Proceedings of the 2nd International Conference on Evolutionary Multi-Criterion Optimization (EMO)*, volume 2632 of *LNCS*, pages 222–236, 2003.

[54] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation*, 6(2):182–197, 2002.

[55] I. M. del Águila and J. del Sagrado. Bayesian networks for enhancement of requirements engineering: a literature review. *Requirements Engineering*, 21(4):461–480, 2016.

[56] N. Desnos, M. Huchard, G. Tremblay, C. Urtado, and S. Vauttier. Search-based many-to-one component substitution. *Journal of Software Maintenance and Evolution*, 20(5):321–344, 2008.

[57] L. Dobrica and E. Niemela. A survey on software architecture analysis methods. *IEEE Transactions on Software Engineering*, 28(7):638–653, 2002.

[58] G. Dodig-Crnkovic. Scientific methods in computer science. In *Proceedings of the Conference for the Promotion of Research in IT at New Universities and at University Colleges in Sweden*, pages 126–130, 2002.

[59] S. Ducasse and D. Pollet. Software Architecture Reconstruction: A Process-Oriented Taxonomy. *Journal of Transactions on Software Engineering*, 35(4):573–591, 2009.

[60] A. Dutta. Integrating AI and optimization for decision support: a survey. *Decision Support Systems*, 18(3):217–226, 1996.

[61] S. Easterbrook, J. Singer, M.-A. Storey, and D. Damian. Selecting Empirical Methods for Software Engineering Research. In *Guide to Advanced Empirical Software Engineering*, chapter 11, pages 285–311. Springer London, London, 2008.

[62] A. E. Eiben and J. E. Smith. *Introduction to Evolutionary Computing*. Springer-Verlag Berlin Heidelberg, 2nd edition, 2015.

[63] R. Etemaadi, K. Lind, R. Heldal, and M. R. V. Chaudron. Quality-driven optimization of system architecture: Industrial case study on an automotive sub-system. *Journal of Systems and Software*, 86(10):2559–2573, 2013.

[64] D. Falessi, G. Cantone, R. Kazman, and P. Kruchten. Decision-making techniques for software architecture design. *ACM Computing Surveys*, 43(4):1–28, 2011.

[65] Y.-Y. FanJiang and Y. Syu. Semantic-based automatic service composition with functional and non-functional requirements in design time: A genetic algorithm approach. *Information and Software Technology*, 56(3):352–373, 2014.

[66] R. Feldt and A. Magazinius. Validity Threats in Empirical Software Engineering Research - An Initial Survey. In *Proceedings of the 22nd International Conference on Software Engineering and Knowledge Engineering (SEKE)*, pages 374–379, 2010.

[67] T. N. Ferreira, S. R. Vergilio, and J. T. de Souza. Incorporating User Preferences in Search-Based Software Engineering: A Systematic Mapping Study. *Information and Software Technology*, 90:55–69, 2017.

[68] F. Ferruci, M. Harman, and F. Sarro. Search-Based Software Project Management. In *Software Project Management in a Changing World*, pages 373–399. Springer-Verlag Berlin Heidelberg, 2014.

[69] M. Fleck, J. Troya, M. Kessentini, and M. Wimmer. Model Transformation Modularization as a Many-Objective Optimization Problem. *IEEE Transactions on Software Engineering*, 43(11):1009–1032, 2017.

[70] G. Fraser, M. Staats, P. McMinn, A. Arcuri, and F. Padberg. Does automated unit test generation really help software testers? a controlled empirical study. *ACM Transactions on Software Engineering and Methodology*, 24(4):23, 2015.

[71] M. Friedman. The use of ranks to avoid the assumption of normality implicit in the analysis of variance. *Journal of the American Statistical Association*, 32(200):675–701, 1937.

[72] J. Garcia, P. Daniel, G. Edwards, and N. Medvidovic. Identifying Architectural Bad Smells. In *Proceedings of the 13th European Conference on Software Maintenance and Reengineering (CSMR)*, pages 255–258, 2009.

[73] D. Garlan. Software architecture: a roadmap. In *Proceedings of the Conference on The Future of Software Engineering*, pages 91–101, 2000.

[74] M. Gendreau and J.-Y. Potvin, editors. *Handbook of Metaheuristics*, volume 146 of *International Series in Operations Research & Management Science*. Springer US, 2010.

[75] F. Glover. Future paths for integer programming and links to artificial intelligence. *Computers & Operations Research*, 13(5):533–549, 1986.

[76] I. Gorton. *Essential Software Architecture*. Springer, 2nd edition, 2011.

[77] L. Grunske. Identifying "good" architectural design alternatives with multiobjective optimization strategies. In *Proceedings of the 28th International Conference on Software Engineering (ICSE)*, pages 849–852, 2006.

[78] M. Harman. The role of Artificial Intelligence in Software Engineering. In *Proceedings of the 1st International Workshop on Realizing AI Synergies in Software Engineering (RAISE)*, pages 1–6, 2012.

[79] M. Harman, S. Afshin Mansouri, and Y. Zhang. Search Based Software Engineering: Trends, Techniques and Applications. *ACM Computing Surveys*, 45(1):1–64, 2012.

[80] M. Harman and J. Clark. Metrics are fitness functions too. In *Proceedings of the 10th International Symposium on Software Metrics*, pages 58–69, 2004.

[81] M. Harman and B. F. Jones. Search-based software engineering. *Information and Software Technology*, 43(14):833–839, 2001.

[82] M. Harman, K. Lakhotia, J. Singer, D. R. White, and S. Yoo. Cloud engineering is Search Based Software Engineering too. *Journal of Systems and Software*, 86(9):2225–2241, 2013.

[83] M. Harman, S. Mansouri, and Y. Zhang. Search Based Software Engineering: A Comprehensive Analysis and Review of Trends, Techniques and Applications. Technical Report TR-09-03, King's College London, 2009.

[84] M. Harman, P. McMinn, J. De Souza, and S. Yoo. Search based software engineering: Techniques, taxonomy, tutorial. In *Empirical Software Engineering and Verification*, volume 7007 LNCS, pages 1–59. Springer Berlin Heidelberg, 2012.

[85] M. Harman and L. Tratt. Pareto Optimal Search Based Refactoring at the Design Level. In *Proceedings of the 9th Genetic and Evolutionary Computation (GECCO)*, pages 1106–1113, 2007.

[86] E. F. Harrison. A process perspective on strategic decision making. *Management Decision*, 34(1):46–53, 1996.

[87] S. M. H. Hasheminejad and S. Jalili. An evolutionary approach to identify logical components. *Journal of Systems and Software*, 96:24–50, 2014.

[88] S. M. H. Hasheminejad and S. Jalili. CCIC: Clustering analysis classes to identify software components. *Information and Software Technology*, 57:329–351, 2015.

[89] Y. He, X. Wang, and J. Z. Huang. Recent advances in multiple criteria decision making techniques. *International Journal of Machine Learning and Cybernetics*, pages 1–4, 2016.

[90] S. Herold and M. Mair. Recommending refactorings to re-establish architectural consistency. In *Proceedings of the 8th European Conference on Software Architecture (ECSA)*, volume 8627 LNCS, pages 390–397, 2014.

[91] S. Holm. A simple sequentially rejective multiple test procedure. *Scandinavian Journal of Statistics*, 6:65–70, 1979.

[92] International MCDM Society. *Definition of Multiple Criteria Decision Making.* http://www.mcdmsociety.org/.

[93] ISO. *ISO/IEC 25010:2011(E). Software product Quality Requirements and Evaluation (SQuaRE) - System and software quality models*, 2011.

[94] ISO. *ISO/IEC FDIS 42010/D9. Systems and software engineering - Architecture description*, 2011.

[95] W. Jung, E. Lee, and C. Wu. A survey on mining software repositories. *IEICE Transactions on Information and Systems*, E95-D(5):1384–1406, 2012.

[96] S. Kebir, I. Borne, and D. Meslati. A genetic algorithm-based approach for automated refactoring of component-based software. *Information and Software Technology*, 88:17–36, 2017.

[97] S. Kebir, A. D. Seriai, S. Chardigny, and A. Chaoui. Quality-centric approach for software component identification from object-oriented code. In *Proceedings of the 10th Joint Working Conference on Software Architecture and 6th European Conference on Software Architecture (WICSA/ECSA)*, pages 181–190, 2012.

[98] M. Kessentini, A. Ouni, P. Langer, M. Wimmer, and S. Bechikh. Search-based metamodel matching with structural and syntactic measures. *The Journal of Systems & Software*, 97:1–14, 2014.

[99] M. Kessentini, H. Sahraoui, M. Boukadoum, and O. B. Omar. Search-based model transformation by example. *Software and Systems Modeling*, 11(2):209–226, 2012.

[100] M. Kessentini, W. Werda, P. Langer, and M. Wimmer. Search-based model merging. In *Proceedings of the 15th Genetic and Evolutionary Computation Conference (GECCO)*, pages 1453–1460, 2013.

[101] V. Khare, X. Yao, and K. Deb. Performance Scaling of Multi-objective Evolutionary Algorithms. In *Proceedings of the 2nd International Conference on Evolutionary Multi-Criterion Optimization (EMO)*, volume 2632 of *LNCS*, pages 376–390. Springer Berlin Heidelberg, 2003.

[102] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, 1983.

[103] B. Kitchenham and S. Charters. Guidelines for performing Systematic Literature Reviews in Software Engineering. Technical report, Keele University and Durham University, 2007.

[104] B. A. Kitchenham, D. Budgen, and P. Brereton. *Evidence-Based Software Engineering and Systematic Reviews*. CRC Press, 2016.

[105] M. Koksalan and I. Karahan. An Interactive Territory Defining Evolutionary Algorithm: iTDEA. *IEEE Transactions on Evolutionary Computation*, 14(5):702–722, 2010.

[106] M. Köppen and K. Yoshida. Substitute Distance Assignments in NSGA-II for Handling Many-Objective Optimization Problems. In *Evolutionary MultiCriterion Optimization*, pages 727–741, 2007.

[107] A. Koziolek, D. Ardagna, and R. Mirandola. Hybrid multi-attribute QoS optimization in component based software systems. *Journal of Systems and Software*, 86(10):2542–2558, 2013.

[108] N. Krasnogor and J. Smith. A tutorial for competent memetic algorithms: Model, taxonomy, and design issues. *IEEE Transactions on Evolutionary Computation*, 9(5):474–488, 2005.

[109] J. K. Lee, S. J. Jung, S. D. Kim, W. H. Jang, and D. H. Ham. Component identification method with coupling and cohesion. In *Proceedings of the 8th Asia-Pacific Software Engineering Conference (APSEC)*, 2001.

[110] B. Li, J. Li, K. Tang, and X. Yao. Many-Objective Evolutionary Algorithms. *ACM Computing Surveys*, 48(1):1–35, 2015.

[111] R. E. Lopez-Herrejon, L. Linsbauer, and A. Egyed. A systematic mapping study of search-based software engineering for software product lines. *Information and Software Technology*, 61:33–51, 2015.

[112] A. López Jaimes and C. A. Coello Coello. Many-Objective Problems: Challenges and Methods. In *Springer Handbook of Computational Intelligence*, pages 1033–1046. Springer Berlin Heidelberg, 2015.

[113] F. Losavio, L. Chirinos, A. Matteo, N. Lévy, and A. Ramdane-Cherif. ISO quality standards for measuring architectures. *Journal of Systems and Software*, 72(2):209–223, 2004.

[114] A. Lozano-Tello and A. Gómez-Pérez. BAREMO: How to Choose the Appropriate Software Component Using the Analytic Hierarchy Process. In *Proceedings of the 14th International Conference on Software Engineering and Knowledge Engineering (SEKE)*, pages 781–788, 2002.

[115] T. Lutellier, D. Chollak, J. Garcia, L. Tan, D. Rayside, N. Medvidović, and R. Kroeger. Measuring the Impact of Code Dependencies on Software Architecture Recovery Techniques. *IEEE Transactions on Software Engineering*, 44(2):159–181, 2018.

[116] R. Lutz. Evolving Good Hierarchical Decompositions of Complex Systems. *Journal of Systems Architecture*, 47:613–634, 2001.

[117] S. Mahmood, R. Lai, Y. S. Kim, J. H. Kim, S. C. Park, and H. S. Oh. A survey of component based system quality assurance and assessment. *Information and Software Technology*, 47(10):693–707, 2005.

[118] S. Malek, N. Medvidović, and M. Mikic-Rakic. An extensible framework for improving a distributed software system's deployment architecture. *IEEE Transactions on Software Engineering*, 38(1):73–100, 2012.

[119] R. Malhotra. A systematic review of machine learning techniques for software fault prediction. *Applied Soft Computing*, 27:504–518, 2015.

[120] U. Mansoor, M. Kessentini, M. Wimmer, and K. Deb. Multi-view refactoring of class and activity diagrams using a multi-objective evolutionary algorithm. *Software Quality Journal*, 25(2):473–501, 2017.

[121] O. Maqbool and H. Babri. Hierarchical clustering for software architecture recovery. *IEEE Transactions on Software Engineering*, 33(11):759–780, 2007.

[122] B. Marculescu, R. Feldt, R. Torkar, and S. Poulding. Transferring interactive search-based software testing to industry. *Journal of Systems and Software*, 142:156–170, 2018.

[123] T. Mariani, T. Elita Colanzi, and S. Regina Vergilio. Preserving architectural styles in the search based design of software product line architectures. *Journal of Systems and Software*, 115:157–173, 2016.

[124] T. Mariani and S. R. Vergilio. A systematic review on search-based refactoring. *Information and Software Technology*, 83:14–34, 2017.

[125] J. Marquis, E. S. Gel, J. W. Fowler, M. Köksalan, P. Korhonen, and J. Wallenius. Impact of Number of Interactions, Different Interaction Patterns, and Human Evolutionary Multiobjective Optimization Algorithms. *Decision Sciences*, 46(5):981–1006, 2015.

[126] J. N. Martin. Overview of an Emerging Standard on Architecture Evaluation – ISO/IEC 42030. In *Proceedings of the 27th Annual INCOSE International Symposium*, pages 1139–1156, 2017.

[127] P. McMinn. Search-based software test data generation: A survey. *Software Testing Verification and Reliability*, 14(2):105–156, 2004.

[128] I. Meedeniya, A. Aleti, and L. Grunske. Architecture-driven reliability optimization with uncertain model parameters. *Journal of Systems and Software*, 85(10):2340–2355, 2012.

[129] D. Meignan, S. Knust, J.-M. Frayret, G. Pesant, and N. Gaud. A Review and Taxonomy of Interactive Optimization Methods in Operations Research. *ACM Transactions on Interactive Intelligent Systems*, 5(3):1–43, 2015.

[130] Z. Michalewicz and D. B. Fogel. *How to Solve it: Modern Heuristics*. Springer-Verlag Berlin Heidelberg, 2nd edition, 2004.

[131] S. Moaven, J. Habibi, H. Ahmadi, and A. Kamandi. A Decision Support System for Software Architecture-Style Selection. In *Proceedings of the 6th International Conference on Software Engineering Research, Management and Applications (SERA)*, pages 213–220, 2008.

[132] V. L. Narasimhan and B. Hendradjaya. Some theoretical considerations for a suite of metrics for the integration of software components. *Information Sciences*, 177(3):844–864, 2007.

[133] M. Nicoletti, S. Schiaffino, and J. A. Diaz-Pace. An optimization-based tool to support the cost-effective production of software architecture documentation. *Journal of Software: Evolution and Process*, 27:674–699, 2015.

[134] M. Nowak and C. Pautasso. Goals, Questions and Metrics for Architectural Decision Models. In *Proceedings of the 6th international workshop on SHAring and Reusing architectural Knowledge (SHARK)*, pages 21–28, 2011.

[135] M. Ó Cinnéide, I. Hemati Moghadam, M. Harman, S. Counsell, and L. Tratt. An experimental search-based approach to cohesion metric evaluation. *Empirical Software Engineering*, pages 1–38, 2016.

[136] L. O'Brien, L. Bass, and P. Merson. Quality attributes and service-oriented architectures. Technical Report CMU/SEI-2005-TN-014, Carnegie Mellon University, 2005.

[137] M. O'Keeffe and M. Ó Cinnéide. Search-based software maintenance. *Proceedings of the European Conference on Software Maintenance and Reengineering (CSMR)*, pages 249–258, 2006.

[138] OMG. *Unified Modeling Language 2.5.1*, Dec. 2017. https://www.omg.org/spec/UML/2.5.1.

[139] S. Orlov and A. Vishnyakov. Decision Making for the Software Architecture Structure Based on the Criteria Importance Theory. *Procedia Computer Science*, 104:27–34, 2017. ICTE 2016, Riga Technical University, Latvia.

[140] I. H. Osman and G. Laporte. Metaheuristics: A bibliography. *Annals of Operations Research*, 63:513–623, 1996.

[141] A. Ouni, M. Kessentini, H. Sahraoui, K. Inoue, and M. S. Hamdi. Improving multi-objective code-smells correction using development history. *Journal of Systems and Software*, 105:18–39, 2015.

[142] R. Paiva, G. N. Rodrigues, R. Bonifácio, and M. Ladeira. Exploring the Combination of Software Visualization and Data Clustering in the Software Architecture Recovery Process. In *Proceedings of the 31st ACM Symposium on Applied Computing (SAC)*, pages 1309–1314, 2016.

[143] M. P. Papazoglou and W.-J. van den Heuvel. Service oriented architectures: approaches, technologies and research issues. *The VLDB Journal*, 16(3):389–415, 2007.

[144] J. A. Parejo, J. García, A. Ruiz-Cortés, and J. C. Riquelme. STATService: Herramienta de análisis estadístico como soporte para la investigación con Metaheurísticas. In *Proceedings of the VIII Congreso Expañol sobre Metaheurísticas, Algoritmos Evolutivos y Bio-inspirados*, 2012.

[145] J. A. Parejo, S. Segura, P. Fernandez, and A. Ruiz-Cortés. QoS-aware web services composition using GRASP with Path Relinking. *Expert Systems with Applications*, 41(9):4211–4223, jul 2014.

[146] I. C. Parmee. Poor-Definition, Uncertainty, and Human Factors - Satisfying Multiple Objectives in Real-World Decision-Making Environments. In *Proceedings of the 1st International Conference on Evolutionary Multi-Criterion Optimization (EMO)*, pages 52–66, 2001.

[147] D. Partridge. Artificial intelligence and software engineering: a survey of possibilities. *Information and Software Technology*, 30(3):146–152, 1988.

[148] D. Perez-Palacin, R. Mirandola, and J. Merseguer. On the relationships between QoS and software adaptability at the architectural level. *Journal of Systems and Software*, 87(1):1–17, 2014.

[149] J. Petke, S. O. Haraldsson, M. Harman, W. B. Langdon, D. R. White, and J. R. Woodward. Genetic Improvement of Software: A Comprehensive Survey. *IEEE Transactions on Evolutionary Computation*, 22(3):415–432, 2018.

[150] G. Phillips-Wren and N. Ichalkaranje, editors. *Intelligent Decision Making: An AI-Based Approach*. Springer-Verlag Berlin Heidelberg, 1st edition, 2008.

[151] A. M. Pitangueira, R. S. P. Maciel, and M. Barros. Software requirements selection and prioritization using SBSE approaches: A systematic review and mapping of the literature. *Journal of Systems and Software*, 103:267–280, 2015.

[152] K. Praditwong and X. Yao. How well do multi-objective evolutionary algorithms scale to large problems. In *Proceedings of the IEEE Congress on Evolutionary Computation (CEC)*, pages 3959–3966, 2007.

[153] R. S. Pressman. *Software Engineering: A Practitioner's Approach*. McGraw Hill, 8th edition, 2014.

[154] R. Purshouse and P. Fleming. On the evolutionary optimisation of many conflicting objectives. *IEEE Transactions on Evolutionary Computation*, 11(6):770–784, 2007.

[155] R. C. Purshouse and P. J. Fleming. Evolutionary Many-Objective Optimisation: An Exploratory Analysis. In *Proceedings of the IEEE Congress on Evolutionary Computation (CEC)*, pages 2066–2073, 2003.

[156] R. *The R Project for Statistical Computing*, 2018. https://www.r-project.org/.

[157] O. Räihä. *Genetic Algorithms in Software Architecture Synthesis*. PhD thesis, University of Tampere, 2008.

[158] O. Räihä. A survey on search-based software design. *Computer Science Review*, 4(4):203–249, 2010.

[159] O. Räihä and K. Koskimies. Multi-Objective Genetic Synthesis of Software Architecture. In *Proceedings of the Companion Publication of the 13th Genetic and Evolutionary Computation Conference (GECCO Companion)*, pages 249–250, 2011.

[160] O. Räihä, K. Koskimies, and E. Mäkinen. Complementary crossover for genetic software architecture synthesis. In *Proceedings of the 10th International Conference on Intelligent Systems Design and Applications (ISDA)*, pages 266–271, 2010.

[161] M. Ravber, M. Mernik, and M. Črepinšek. The impact of Quality Indicators on the rating of Multi-objective Evolutionary Algorithms. *Applied Soft Computing*, 55:265–275, 2017.

[162] S. Rekha V. and H. Muccini. Suitability of Software Architecture Decision Making Methods for Group Decisions. In *Proceedings of the 8th European Conference on Software Architecture (ECSA)*, pages 17–32, 2014.

[163] P. Rodríguez-Mier, M. Mucientes, M. Lama, and M. I. Couto. Composition of web services through genetic programming. *Evolutionary Intelligence*, 3(3):171–186, 2010.

[164] J. Romano, J. D. Kromrey, J. Coraggio, and J. Showronek. Appropriate statistics for ordinal level data: Should we really be using t-test and cohen's d for evaluating group differences on the nsse and other surveys? In *Annual Meeting of the Florida Association of Institutional Research*, 2006.

[165] J. R. Romero, J. M. Luna, and S. Ventura. datapro4: the data processing library for Java, 2012. http://www.uco.es/grupos/kdis/datapro4j.

[166] S. Russell and P. Norvig. *Artiticial Intelligence: A Modern Approach*. Prentice Hall, 3rd edition, 2016.

[167] C. Sant'Anna, E. Figueiredo, A. Garcia, and C. J. P. Lucena. On the Modularity of Software Architectures: A Concern-Driven Measurement Framework. In *Proceedings of the 1st European Conference on Software Architecture (ECSA)*, pages 207–224, 2007.

[168] G. R. Santhanam. Qualitative optimization in software engineering: A short survey. *Journal of Systems and Software*, 111:149–156, 2016.

[169] A. S. Sayyad and H. Ammar. Pareto-optimal search-based software engineering (POSBSE): A literature survey. In *Proceedings of the 2nd International*

Workshop on Realizing Artificial Intelligence Synergies in Software Engineering (RAISE), pages 21–27, 2013.

[170] J. D. Schaffer. Multiple Objective Optimization with Vector Evaluated Genetic Algorithms. In *Proceedings of the 1st International Conference on Genetic Algorithms*, pages 93–100, 1985.

[171] O. Schütze, A. Lara, and C. A. Coello Coello. On the influence of the number of objectives on the hardness of a multiobjective optimization problem. *IEEE Transactions on Evolutionary Computation*, 15(4):444–455, 2011.

[172] SDMetrics. *SDMetrics core functionality*, 2018. https://www.sdmetrics.com/OpenCore.html.

[173] A.-D. Seriai and S. Chardigny. A Genetic Approach for Software Architecture Recovery from Object-Oriented Code. In *Proceedings of the 23rd International Conference on Software Engineering & Knowledge Engineering (SEKE)*, pages 515–520, 2011.

[174] M. R. N. Shackelford. Implementation issues for an interactive evolutionary computation system. In *Proceedings of Companion Publication of the 9th Genetic and Evolutionary Computation Conference (GECCO)*, pages 2933–2935, 2007.

[175] M. R. N. Shackelford and C. L. Simons. Metaheuristic Design Pattern: Interactive Solution Presentation. In *Proceedings of the Companion Publication of the 16th Genetic and Evolutionary Computation Conference (GECCO)*, pages 1431–1433, 2014.

[176] Q. Z. Sheng, X. Qiao, A. V. Vasilakos, C. Szabo, S. Bourne, and X. Xu. Web services composition: A decade's overview. *Information Sciences*, 280:218–238, 2014.

[177] O. Sievi-Korte, E. Mäkinen, and T. Poranen. Simulated Annealing for Aiding Genetic Algorithm in Software Architecture Synthesis. *Acta Cybernetica*, 21(2):235–265, 2013.

[178] C. L. Simons. *Interactive Evolutionary Computing in Early Lifecycle Software Engineering Design*. PhD thesis, University of the West of England, 2011.

[179] C. L. Simons. Whither (away) Software Engineers in SBSE? In *Proceedings of the 1st International Workshop on Combining Modelling and Search-Based Software Engineering (CMSBSE)*, pages 49–50, 2013.

[180] C. L. Simons and I. C. Parmee. Elegant Object-Oriented Software Design via Interactive, Evolutionary Computation. *IEEE Transactions on Systems, Man and Cybernetics Part C: Applications and Reviews*, 42(6):1797–1805, 2012.

[181] C. L. Simons, I. C. Parmee, and R. Gwynllyw. Interactive, Evolutionary Search in Upstream Object-Oriented Class Design. *IEEE Transactions on Software Engineering*, 36(6):798–816, 2010.

[182] C. L. Simons, J. Smith, and P. White. Interactive ant colony optimization (iACO) for early lifecycle software design. *Swarm Intelligence*, 8(2):139–157, 2014.

[183] I. Stavropoulou, M. Grigoriou, and K. Kontogiannis. Case study on which relations to use for clustering-based software architecture recovery. *Empirical Software Engineering*, 22(4):1717–1762, 2017.

[184] M. Svahnberg, C. Wohlin, L. Lundberg, and M. Mattson. A Quality-Driven Decision Support Method for Identifying Software Architecture Candidates. *International Journal of Software Engineering and Knowledge Engineering*, 13(05):547–573, 2003.

[185] C. Szyperski. *Component Software: Beyond Object-Oriented Programming*. Addison-Wesley Longman, 2nd edition, 2002.

[186] H. Takagi. Interactive evolutionary computation: fusion of the capabilities of EC optimization and human evaluation. *Proceedings of the IEEE*, 89(9):1275–1296, 2001.

[187] E.-G. Talbi. *Metaheuristics: From Design to Implementation*. Wiley, 2009.

[188] D. A. Tamburri and R. Kazman. General methods for software architecture recovery: a potential approach and its evaluation. *Empirical Software Engineering*, 23(3):1457–1489, 2018.

[189] A. Tosun, A. B. Bener, and S. Akbarinasaji. A systematic literature review on the applications of Bayesian networks to predict software quality. *Software Quality Journal*, 25(1):273–305, 2017.

[190] E. Triantaphyllou. *Multi-criteria Decision Making Methods: A Comparative Study*. Number 44 in Applied Optimization. Springer, 1st edition, 2000.

[191] I. Trummer, B. Faltings, and W. Binder. Multi-Objective Quality-Driven Service Selection - A Fully Polynomial Time Approximation Scheme. *IEEE Transactions on Software Engineering*, 40(2):167–191, 2014.

[192] S. Vathsavayi, Hadaytullah, and K. Koskimies. Interleaving human and search-based software architecture design. *Proceedings of the Estonian Academy of Sciences*, 62(1):16–26, 2013.

[193] S. Ventura, C. Romero, A. Zafra, J. A. Delgado, and C. Hervás. JCLEC: A Java framework for evolutionary computation. *Soft Computing*, 12(4):381–392, 2008.

[194] T. Vernazza, G. Succi, and G. Granatella. Defining Metrics for Software Components. In *Proceedings of the World Multiconference on Systemics, Cybernetics and Informatics*, pages 16–23, 2000.

[195] A. Vescan and C. Şerban. Multilevel component selection optimization toward an optimal architecture. *Soft Computing*, 21(15):4481–4495, 2017.

[196] C. Von Lücken, B. Barán, and C. Brizuela. A survey on multi-objective evolutionary algorithms for many-objective problems. *Computational Optimization and Applications*, 58(3):707–756, 2014.

[197] H. Wada, J. Suzuki, Y. Yamano, and K. Oba. E3: A Multiobjective Optimization Framework for SLA-Aware Service Composition. *IEEE Transactions on Services Computing*, 5(3):358–372, 2012.

[198] T. Wagner, N. Beume, and B. Naujoks. Pareto-, aggregation-, and indicator-based methods in many-objective optimization. In *Evolutionary MultiCriterion Optimization*, volume 4403, pages 742–756, 2007.

[199] D. J. Walker, R. M. Everson, and J. E. Fieldsend. Visualizing mutually non-dominating solution sets in many-objective optimization. *IEEE Transactions on Evolutionary Computation*, 17(2):165–184, 2013.

[200] S. Wang, S. Ali, T. Yue, Y. Li, and M. Liaaen. A practical guide to select quality indicators for assessing pareto-based search algorithms in search-based software engineering. *Proceedings of the 38th International Conference on Software Engineering (ICSE)*, pages 631–642, 2016.

[201] H. Washizaki, H. Yamamoto, and Y. Fukazawa. A metrics suite for measuring reusability of software components. In *Proceedings of the 5th International Workshop on Enterprise Networking and Computing in Healthcare Industry*, pages 211–223, 2003.

[202] J. Wen, S. Li, Z. Lin, Y. Hu, and C. Huang. Systematic literature review of machine learning based software development effort estimation models. *Information and Software Technology*, 54(1):41–59, 2012.

[203] F. Wilcoxon. Individual Comparison by Ranking Methods. *Biometrics*, 1:80–83, 1945.

[204] Y. Xu and P. Liang. Automated Software Architectural Synthesis using Patterns: A Cooperative Coevolution Approach. *Proceedings of the 26th International Conference on Software Engineering and Knowledge Engineering*, 24(10):1387–1411, 2014.

[205] S. Yang, M. Li, X. Liu, and J. Zheng. A Grid-Based Evolutionary Algorithm for Many-Objective Optimization. *IEEE Transactions on Evolutionary Computation*, 17(5):721–736, 2013.

[206] Y. Yu, H. Ma, and M. Zhang. F-MOGP: A novel many-objective evolutionary approach to QoS-aware data intensive web service composition. In *Proceedings of the IEEE Congress on Evolutionary Computation (CEC)*, pages 2843–2850, 2015.

[207] Q. Zhang and H. Li. MOEA/D: A multiobjective evolutionary algorithm based on decomposition. *IEEE Transactions on Evolutionary Computation*, 11(6):712–731, 2007.

[208] Y. Zhang, M. Harman, and A. Mansouri. *The SBSE Repository: A repository and analysis of authors and research articles on Search Based Software Engineering*. http://crestweb.cs.ucl.ac.uk/resources/sbse_repository/.

[209] A. Zhou, B.-Y. Qu, H. Li, S.-Z. Zhao, P. N. Suganthan, and Q. Zhang. Multiobjective evolutionary algorithms: A survey of the state of the art. *Swarm and Evolutionary Computation*, 1(1):32–49, 2011.

[210] E. Zitzler, K. Deb, and L. Thiele. Comparison of multiobjective evolutionary algorithms: Empirical results. *Evolutionary Computation*, 8(2):173–195, 2000.

[211] E. Zitzler, M. Laumanns, and S. Bleuler. A Tutorial on Evolutionary Multiobjective Optimization. In *Metaheuristics for Multiobjective Optimisation*, volume 535 of *LNE*, pages 3–37. Springer Berlin Heidelberg, 2004.

[212] E. Zitzler, M. Laumanns, and L. Thiele. SPEA2: Improving the Strength Pareto Evolutionary Algorithm. In *Proceedings of the International Conference on Evolutionary Methods for Design Optimization and Control with Applications to Industrial Problems (EUROGEN)*, pages 95–100, 2001.

[213] E. Zitzler and K. Simon. Indicator-Based Selection in Multiobjective Search. In *Proceedings of the 8th International Conference on Parallel Problem Solving from Nature (PPSN)*, pages 832–842, 2004.

Scientific publications

[C1] A. Ramírez, J. R. Romero, and S. Ventura. On the Performance of Multiple Objective Evolutionary Algorithms for Software Architecture Discovery. *Proceedings of the 16th Genetic and Evolutionary Computation Conference (GECCO)*, pp. 1287–1294, 2014.

[C2] A. Ramírez, J. R. Romero, and S. Ventura. Análisis de la aplicabilidad de medidas software para el diseño semi-automático de arquitecturas. *Proceedings of the XIX Jornadas en Ingeniería del Software y Bases de Datos (JISBD)*, pp. 307–320, 2014.

[C3] A. Ramírez, J. R. Romero, and S. Ventura. Estudio preliminar del rendimiento de familias de algoritmos multiobjetivo en diseño arquitectónico. *Proceedings of the X Congreso Español sobre Metaheurísticas, Algoritmos Evolutivos y Bioinspirados (MAEB)*, pp. 173–180, 2015.

[C4] A. Ramírez, J. R. Romero, and S. Ventura. An Extensible JCLEC-based Solution for the Implementation of Multi-Objective Evolutionary Algorithms. *Proceedings of the Companion Publication of 17th Genetic and Evolutionary Computation Conference (GECCO Companion)*, pp. 1085–1092, 2015.

[C5] A. Ramírez, J. R. Romero, and S. Ventura. Interactividad en el descubrimiento evolutivo de arquitecturas. *Proceedings of the XX Jornadas en Ingeniería del Software y Bases de Datos (JISBD)*, 2015.

[J1] A. Ramírez, J. R. Romero, and S. Ventura. An approach for the evolutionary discovery of software architectures. *Information Sciences*, vol. 305, pp. 234–255, 2015.

[C6] A. Ramírez, J. A. Molina, J. R. Romero, and S. Ventura. Estudio de mecanismos de hibridación para el descubrimiento evolutivo de arquitecturas. *Proceedings of the XXI Jornadas en Ingeniería del Software y Bases de Datos (JISBD)*, pp. 481–494, 2016.

[C7] J. Parejo, A. Ramírez, J. Romero, S. Segura, and A. Ruiz-Cortés. Configuración guiada por búsqueda de aplicaciones basadas en microservicios en la nube. *Proceedings of the XXI Jornadas en Ingeniería del Software y Bases de Datos (JISBD)*, pp. 499–502, 2016.

[C8] A. Ramírez, R. Barbudo, J. R. Romero, and S. Ventura. Herramienta basada en computación evolutiva interactiva para arquitectos software. *Proceedings of the XI Congreso Español sobre Metaheurísticas, Algoritmos Evolutivos y Bioinspirados (MAEB)*, pp. 387–396, 2016.

[C9] A. Ramírez, R. Barbudo, J. R. Romero, and S. Ventura. Memetic Algorithms for the Automatic Discovery of Software Architectures. *Proceedings of the 16th International Conference on Intelligent Systems Design and Applications (ISDA)*, vol. 557 AISC, pp. 437–447, 2016.

[J2] A. Ramírez, J. R. Romero, and S. Ventura. A comparative study of many-objective evolutionary algorithms for the discovery of software architectures. *Empirical Software Engineering*, vol. 21, no. 6, pp. 2546–2600, 2016.

[C10] A. Ramírez, J. R. Romero, and S. Ventura. On the effect of local search in the multi-objective evolutionary discovery of software architectures. *Proceedings of the IEEE Congress on Evolutionary Computation (CEC)*, pp. 2038–2045, 2017.

[C11] A. Ramírez, J. R. Romero, and S. Ventura. Búsqueda coevolutiva interactiva aplicada al diseño de software. *Proceedings of the XXII Jornadas en Ingeniería del Software y Bases de Datos (JISBD)*, 2017.

[J3] A. Ramírez, J. A. Parejo, J. R. Romero, S. Segura, and A. Ruiz-Cortés. Evolutionary composition of QoS-aware web services: A many-objective perspective. *Expert Systems with Applications*, vol. 72, pp. 357–370, 2017.

[J4] A. Ramírez, J. R. Romero, and C. Simons. A Systematic Review of Interaction in Search-Based Software Engineering. *IEEE Transactions on Software Engineering*, pp. 1–22, 2018. *In press.*

[C12] A. Ramírez, J. R. Romero, and S. Ventura. API para el desarrollo de algoritmos interactivos en ingeniería del software basada en búsqueda. *Proceedings of the XXIII Jornadas en Ingeniería del Software y Bases de Datos (JISBD)*, 2018.

[J5] A. Ramírez, J. R. Romero, and S. Ventura. Interactive multi-objective optimization of software architectures. *Information Sciences*, vol. 463-464, pp. 92-109, 2018.

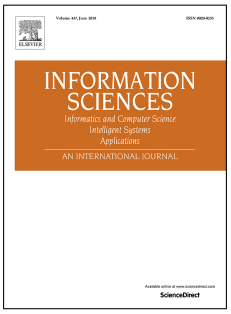
Part II

Scientific Publications

6

Compendium of publications

6.1. An approach for the evolutionary discovery of software architectures



Title	An approach for the evolutionary discovery of software architectures
Authors	A. Ramírez, J.R. Romero, S. Ventura
Journal	Information Sciences
Volume	305
Pages	234-255
Year	2015
Editorial	Elsevier
DOI	10.1016/j.ins.2015.01.017

IF (JCR 2015)	3.364
Category	Computer Science, Information Systems
Position	8/144 (Q1)
Cites	7 (WoS), 11 (Scopus)

Information Sciences 305 (2015) 234–255

Contents lists available at ScienceDirect

Information Sciences

journal homepage: www.elsevier.com/locate/ins

An approach for the evolutionary discovery of software architectures

Aurora Ramírez, José Raúl Romero *, Sebastián Ventura

Department of Computer Science and Numerical Analysis, University of Córdoba, 14071 Córdoba, Spain

ARTICLE INFO

Article history:
Received 30 July 2014
Received in revised form 6 January 2015
Accepted 24 January 2015
Available online 7 February 2015

Keywords:
Search based software engineering
Software architecture discovery
Evolutionary algorithms
Ranking aggregation fitness

ABSTRACT

Software architectures constitute important analysis artefacts in software projects, as they reflect the main functional blocks of the software. They provide high-level analysis artefacts that are useful when architects need to analyse the structure of working systems. Normally, they do this process manually, supported by their prior experiences. Even so, the task can be very tedious when the actual design is unclear due to continuous uncontrolled modifications. Since the recent appearance of search based software engineering, multiple tasks in the area of software engineering have been formulated as complex search and optimisation problems, where evolutionary computation has found a new area of application. This paper explores the design of an evolutionary algorithm (EA) for the discovery of the underlying architecture of software systems. Important efforts have been directed towards the creation of a generic and human-oriented process. Hence, the selection of a comprehensible encoding, a fitness function inspired by accurate software design metrics, and a genetic operator simulating architectural transformations all represent important characteristics of the proposed approach. Finally, a complete parameter study and experimentation have been performed using real software systems, looking for a generic evolutionary approach to help software engineers towards their decision making process.

1. Introduction

Throughout software development, software engineers need to make decisions about the most appropriate structures, platforms and styles of their designs. The automatic inference and evaluation of different design alternatives is a challenging application domain where computational intelligence techniques serve to provide support to software engineers, especially when limited information about the system being developed is still available.

In this context, architectural analysis constitutes an important phase in software projects, as it provides methods and techniques for handling the specification and design of software in the earlier stages [9]. It is considered a human-centered decision process with a great impact on the quality and reusability of the end product. During high level analysis, component identification allows the discovery of system blocks, their functionalities and interactions. For this reason, it is a good practice when dealing with complex systems [35], resulting in more comprehensible software and making its development and maintenance simpler and more affordable.

Frequently, software engineers need to tackle architectural analysis from a working system in order to migrate it or extend its functionality [10]. This could be a difficult task when the underlying system conception has been perverted

* Corresponding author. Tel.: +34 957 21 26 60.
E-mail address: jrromero@uco.es (J.R. Romero).

http://dx.doi.org/10.1016/j.ins.2015.01.017
0020-0255/

due to requirements changes. A more dramatic situation occurs when reverse engineering techniques from source code are the only way to extract system information, leading to inappropriate abstractness because of missing documentation. In these cases, engineers must expend their time and effort, with their own experience as their only guarantee, in the manual discovery of these functional blocks.

Architectural optimisation methods in the field of software engineering (SE) have often proposed guidelines and recommendations to modellers for the identification and improvement of software architectures [5,6]. Hence, semi-automatic tools and intelligent systems might be an efficient solution to support the engineering work in order to obtain quality models.

More specifically, the discovery of the architecture of a software specification can also be formulated as the search of the most appropriate distribution of available software artefacts in more abstract units of construction. Traditionally, proposed approaches are based on the refactoring of source code [21,34], implying that architectural blocks are recovered at the end of the development process without regarding analysis decisions. Besides, it is frequent that source code is evolved without an exhaustive control from the analysis perspective, and it is likely not to be representative of the original conception of the system. Instead, the discovery process can be carried out using earlier available information, like the detailed analysis models in the form of class diagrams. These models offer an intermediate view of the software, between the abstractness of the architecture specification and the specificity of the code.

Recently, the combination of metaheuristic approaches and software engineering as problem domain, denominated search based software engineering (SBSE), has undergone a huge growth [17]. Since the appearance of SBSE, evolutionary computation (EC) has emerged as the most applied metaheuristic [16], demonstrating that it constitutes an interesting and complementary way to help software engineers in the improvement of their object-oriented class designs [33] or user interfaces [36]. In this paper, EC is explored as a search technique to extract the underlying software architecture of a system. It constitutes a novelty in SBSE, where architectural discovery has been viewed as a re-engineering task from source code, which is more oriented towards maintenance and refactoring purposes. The main research questions posed in this work are the following:

RQ1: Can single-objective evolutionary algorithms (EA) help the software engineer to identify an initial candidate architecture of a system at a high level of abstraction? Such an approach should be heavily oriented towards the expert domain, looking for the interoperability with software engineering standards and tools, as well as for the comprehension of the elements involved within the evolutionary model.

RQ2: How does the configuration of the algorithm influence both the evolutionary performance and the quality of the returned solution? In order to answer to this question, an in-depth parameter study is required, aiming to provide useful guidelines on this regard to the software architect.

In the proposed evolutionary approach, class diagrams constitute the source artefacts used to abstract the software architecture, which is encoded using a flexible tree structure. Design alternatives are explored by a specific genetic operator applying domain knowledge. Concepts like cohesion and coupling guide the search, defining a ranking-based fitness function.

The rest of the paper is structured as follows. Section 2 introduces some background in SBSE and architectural modelling. Section 3 details the problem description, whereas the evolutionary model is described in Section 4. Next, experimentation is presented in Section 5, including a detailed parameter study. An illustrative example of the approach is explained in Section 6, and results are discussed in Section 7. Finally, concluding remarks are outlined in Section 8.

2. Background

This section presents the most relevant subjects and background related to our work. More specifically, it introduces evolutionary computation as a technique to solve software engineering tasks, as well as the main terminology related to architectural analysis. Finally, previous works on software architecture optimisation in SBSE are presented.

2.1. Evolutionary computation in software engineering

Evolutionary computation [7] is one of the first population-based and bio-inspired metaheuristics for the resolution of optimisation problems. For this reason, EC has been applied for many years now to a variety of topics and considerable efforts have been applied in order to propose new techniques and operators [38] to solve complex applications.

Applying metaheuristics like EC to any domain requires that the scenario to be solved must be reformulated as an optimisation problem. Software engineering is not an exception [8]. Since the appearance of SBSE, considerable efforts have been devoted to this field. Although the first and most studied area has been the automation of test case generation [12], other tasks related to the rest of phases in the software development, from requirements specification [39] to software verification [27], have already been studied. The advances in the field demonstrate that the application of EC to software enhancement is not only focused in the generation of automated programs, other activities classically performed by humans present new challenges.

Since SE is mainly a human-centered activity, the automation of the expert's reasoning presents a great challenge, especially in the analysis and design phases [29]. Design tasks considered in SBSE encompass problems like the conception of both object-oriented [33] and service-oriented [30] architectures, software module clustering [28] or software refactoring

[20]. These activities are characterised by the need of constructing some type of software model from requirements. Both module clustering [28] and software refactoring [20] are more relevant to software maintenance, since existing software artefacts must be scrutinised in order to provide design alternatives.

In [33], an evolutionary algorithm is combined with software agents to extract the most fitting UML class diagram for a given set of methods and attributes from use cases. This type of software requirement information is also taken as an input of the evolutionary approach proposed in [18], where logical groups of use cases are identified and put together into component packages. In this case, the authors presented a generic framework inspired by clustering techniques. In [30], genetic programming is used to deal with service composition in order to obtain the best orchestration of web services.

Frequently, popular evolutionary schemes and generic operators are selected and adapted when needed, since EC research history provides sufficient candidate elements [22]. Quite the opposite occurs when addressing more complex problems and specific implementations are required [11]. The problem description determines the need for either generic or specific elements. In this sense, the genetic algorithms conceived in [18,28] handle integer encodings for the allocation of software artefacts, whereas those designed in [20,30] require tree structures and special operators for the application of genetic programming approaches. Additionally, an object-oriented encoding to represent the set of classes, methods and attributes is proposed in [33].

2.2. Component-based software architectures

According to the ISO Std. 42010 [19], the architecture of a software system conceives "the fundamental concepts or properties of a system in its environment embodied in its elements, relationships, and in the principles of its design and evolution". These high-level abstraction models are very appropriate for guiding and controlling its subsequent development, since it constitutes a bridge between software requirements and the implementation with specific programming languages [13]. Ideally, these models should be considered during the entire process, evolving as the software does. However, they frequently tend to be shifted during the implementation and the maintenance phases, since architectures do not have a direct representation in code artefacts.

A component-based architecture depicts a special type of architectural model, founded on the idea of constructing the software by means of independent artefacts that aim to promote the reuse of functionality. A commonly accepted definition of *component* is given by Szyperski [35]: "a component is a unit of composition with contractually specified interfaces and explicit context dependencies only". It is also mentioned that a component "can be deployed independently and is subject to third party composition". Internal objects implement its functionality, although they remain hidden and inaccessible beyond the component limits. Relations between components should be defined by means of provided and/or required interfaces. A *provided interface* is defined as "a set of named operations that can be invoked by clients", separating the specification of the functionality from its real implementation. On the other hand, a *required interface* specifies the services invoked by the component, and provided by others. Finally, *connectors* link two or more interaction points between interfaces.

The importance of component-based architectures lies in its capability to represent a variety of software systems by means of abstract units of construction, without the consideration of the final specific technology or context in which the software will be deployed. Thus, components could be implemented as packages, modules or even single classes in object-oriented systems, as well as a set of services provided in the cloud or distributed objects in open and large distributed systems.

Software architects require, just like metaheuristics do, mechanisms to evaluate the adequacy of their models. The most frequently used measures are related to cohesion and coupling [14]. Cohesion refers to the degree to which the elements comprising the component are necessary and sufficient to carry out a single, well-defined function. Coupling is related to the interdependence between components, probably caused by references to other modules and data flows. Good component-based designs, i.e. specifying cohesive components with low dependencies, provide highly scalable software systems with better encapsulation and modularity. There exist other diverse metrics that help engineers in either the measurement of non-functional properties of its component-based designs, like integration [1] or usability [4], or the quantification of the symmetrical elegance of the software design [32].

2.3. Search-based architectural design

Current SBSE proposals in the context of software architectures can be viewed from diverse perspectives [2], since there are different kinds of factors to be considered in an architectural specification. The architecture definition (i.e. modelling languages, available constructs or the algorithm representation); the quality attributes to be measured, either functional or non-functional; the presence of prior components or design patterns, as well as the design purpose (recovery, refactoring, software implementation and hardware deployment) present a variety of application problems.

The conception of a low-level architecture from use cases as a software architecture synthesis problem is proposed in [31]. The authors present a genetic algorithm that takes as input an initial grouping of classes obtained by a simulated annealing algorithm from a graph of functional dependencies, previously extracted from use cases. The resulting architectural specification is composed by classes and interfaces, according to the design patterns that best fit the requirements.

Next, the process proposed in [21] combines a clustering approach and a genetic algorithm for the recovery of component-based architectures from source code. It is a re-engineering model where the genetic search is performed over an initial

architecture obtained after the study of relationships among source code elements (classes, interfaces, packages) from a functional dependency graph. It requires a complex mapping process, since it considers a fixed linear encoding representing the distribution of each class into a component, and needs a transformation mechanism to properly present the solution to the user. The proposed evolutionary model focuses on the architecture reconstruction from source code, its goal being closest to software maintenance. The software code constitutes a powerful source of detailed information about how the system works, but it is not clear that high-level characteristics can be directly extracted from it, since human decisions and abstract information are commonly faded away throughout the software construction process.

Assembling COTS (*components off-the-shelf*) is another example of architecture construction. These COTS already implement specific functionalities whose combination is optimised in order to conform with the overall system functional requirements. In [24], well-known multi-objective genetic algorithms are used to generate design alternatives from an initial component-based architectural model. Besides, the proposal requires precise annotations of the evaluable metrics on each component from the expert, i.e. cost or performance, being components considered as black-box artefacts. Along the same lines, the authors explore in [3] the selection of the optimal subset of pre-existing components, which determine the next release of a system, using simulated annealing and greedy algorithms.

Finally, the framework presented in [23] addresses the issue concerned to architectural deployment, where software components within a distributed system must be allocated in hardware nodes in order to properly satisfy the non-functional requirements given, such as cost, latency and memory consumption. Here, the software specification already exists, so the evolutionary search is focused on exploring several different platforms where the deployed software would be executed.

3. Problem description

The identification of the architectural models is considered during the early stages of software conception, when software modellers still want to modify their current software structure as requirements change or they are requested to check the correctness of the resulting design.

When source code artefacts are not yet available, architects require other sources of information in order to discover the intended architecture. Initial class diagrams, usually the most used representations in the analysis phase, constitute an interesting starting point for architecture discovery. These diagrams offer more specific analysis information than source code, and they use modelling languages like UML 2 [26] instead of programming languages.

Therefore, the originally intended elements that conform a component-based architecture (components, interfaces and connectors) will be identified from these analysis models, resulting in an architecture represented with a UML 2 component diagram. At this point, the semi-automatic discovery of components including its internal structure, candidate interfaces and connectors can be constrained by the following assumptions:

1. A component is defined as a cohesive group of classes, meaning that they work together to satisfy the expected behaviour of the component. Thus, classes within the diagram will be organised searching the best abstraction of the different functionalities that can be identified in the software.
   A very important constraint to consider is that *any class in the input diagram must be contained in one and only one component in the resulting architecture*. Additionally, *any operation or transformation of the architecture must ensure that no empty components are returned*.
2. A directed relationship between classes in the analysis model belonging to different components represents a candidate interface. Although groups of related classes should be allocated in the same component, some interactions could remain between classes belonging to other components, representing operational flows among them. Then, these relationships, required to perform the overall functionality of the system, will be abstracted as interactions between components, i.e. defining its interfaces.
   It can be observed that not all the relationships can constitute a candidate interface. For instance, generalisations represent data abstractions, so they do not imply a flow of operational information. The navigability of the relationship is also important because, if it is not explicitly represented, it would mean that information is exchanged in both directions, the corresponding classes being highly dependent. If the navigability is presented for only one direction, the flow represents a provided or required candidate service.
   Focusing on the interactions between components, *isolated components are not appropriated as they do not provide any service to others*. Secondly, *mutually dependent components are not permitted* from the architectural perspective. This latter circumstance occurs when a component requires and provides services from another component.
3. Connectors can be described as the linkage between a pair of required/provided interfaces interconnecting different components. They will be identified after the discovery of the interfaces created between components.

4. Proposed model for architecture discovery

In this section, the different elements of the proposed evolutionary model are presented, including the encoding chosen, the fitness function and the genetic operator. All these elements are conceived with the aim of creating a comprensible EA as posed by *RQ1*. Finally, the description of the evolutionary algorithm is detailed.

4.1. Encoding of solutions

Selecting the most appropriate problem encoding is a key step in any search algorithm. Usually, a trade-off between the performance and comprehensibility must be achieved, especially when genetic algorithms are aimed at supporting non expert users in metaheuristics. Although the linear encodings proposed in Sections 2.1 and 2.3 seem to be efficient representations, difficult design problems still require its adaptation by means of superstructures or groups of consecutive genes to represent more complex features. In these cases, efficiency decreases due to the use of operators which are too specific or the need for corrective procedures after the application of generic operators.

Human interpretation is usually hampered by complex genotype/phenotype mappings. Therefore, an easier mapping process for software design problems might be beneficial. Tree structures seem to be an interesting option, as they have been used successfully in both computational and human domains. Moreover, these types of representation are also familiar to software architects, because they are common structures in modelling tools, and they allow a flexible management of solutions with different sizes, e.g. architectures with a variable number of components and connectors.

Components, interfaces, connectors and inner elements clearly present a hierarchical composition. Classes and their relationships may constitute a component, whose complete specification requires the definition of its provided and required interfaces. Connectors can be split into the interfaces they link. Then, mapping a component diagram into a tree structure is feasible as shown in Fig. 1, where shading nodes constitute the solution frame, comprised by those mandatory artefacts appearing in any architectural model. The rest of nodes represent the elements that can be different from one solution to another, i.e. a number of component and connectors as well as the distribution of classes and interfaces among them. More specifically, the root node, *Architecture*, represents the component diagram that is comprised of a set of components and connectors. Each component is defined by a node *Component* in terms of its internal classes and its interfaces. Similarly, each connector is described by the pair of required and provided interfaces that it links. Since they are compound elements, they are represented as non-terminal nodes. Finally, classes and interfaces constitute the terminal nodes.

4.2. Initial population

From the problem description (see Section 3), it can be noted that the search space is constituted by all possible combinations of class distribution among components, also identifying its interfaces and the connectors. These candidate groups of classes, and the way in which interfaces and connectors are deduced from them, must also guarantee that the correspondent architecture represents a valid solution.

Firstly, a random number of components is selected between a minimum and a maximum. Default values are set to a minimum of two and a maximum of n components, n being the number of classes in the input model. The higher limit guarantees that no empty components will be generated. Then, each class is assigned to one component, assuring that each component has at least one class. After this initial assignment, the rest of the constraints detailed in Section 3 are omitted, allowing a faster initialisation process. As will be explained later, the main idea is that these invalid individuals will be progressively removed along the generations.

4.3. Ranking fitness function

As mentioned in Section 2.2, diverse functional or non-functional properties can be considered depending on the underlying goal of the architectural optimisation. In this case, the search process is mainly focused on structural aspects, closely related to reusability, since it looks for the optimal identification of well-defined components, interfaces and connectors. Thus, the fitness function considers the strength and independence of the inner functionality of each component.

The fitness function is calculated as an aggregation of rankings. The use of rankings cancels out the need for standardisation between metrics, which could result in an artificial procedure when they are not defined in an appropriate range

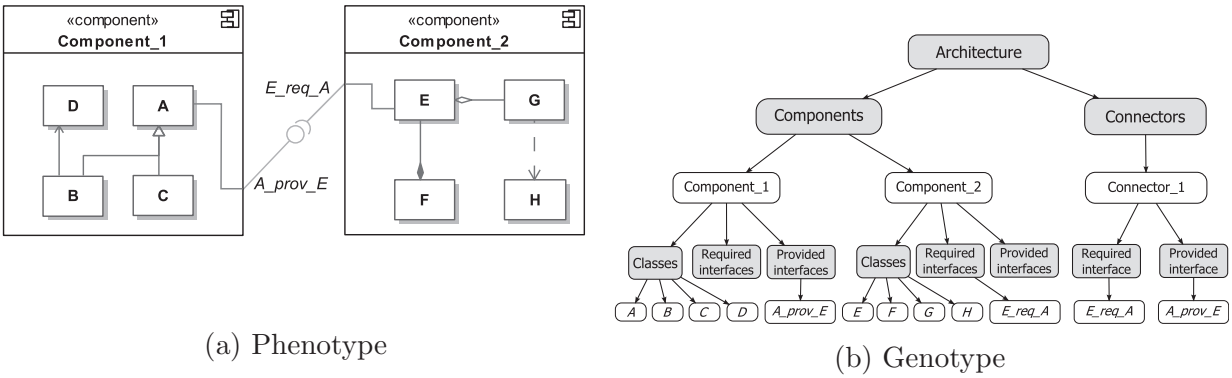


Fig. 1. The phenotype/genotype mapping process.

for a fair scalarisation and aggregation. Each ranking belongs to a specific metric related to desirable characteristics in the architectural design. Therefore, evaluating these design criteria requires the existence of quantifiable measures applicable to the problem domain.

Firstly, the *intra-modular coupling density* (*ICD*) [15] in Eq. (1) serves to determine a trade-off between cohesion and coupling. For each component i, ICD_i is calculated as the ratio between internal and external relations, which has to be maximised. CI_i^{in} is the number of interactions inside the component, i.e. the relationships between classes allocated in the same component. CI_i^{out} represents the number of relationships between component i and the others, i.e. the number of candidate interfaces of the component. Then, every value is properly weighted with the ratio of classes that participate in these relationships. Hence, if two components reach the same ratio of interactions, the smaller component, i.e. the one with less inner classes, is preferable meaning that the density of interactions per class is higher. ICD_i varies in $[0,1]$. Finally, the *ICD* of the overall architecture (individual) is calculated as the average of every ICD_i.

$$\begin{aligned} ICD_i &= ((\#classes_{total} - \#classes_i)/\#classes_{total}) \cdot \left(CI_i^{in}/(CI_i^{in} + CI_i^{out})\right) \\ ICD &= \sum_{i=1}^{n} ICD_i/n \end{aligned} \tag{1}$$

The second metric, named *external relations penalty* (*ERP*), applies a penalty if some relations are not specified by means of interfaces. The optimum value is 0, meaning that no relationship outside the identification of a candidate interface is presented between classes allocated in different components. The minimisation of these dependencies between components is an important characteristic to be considered, as it reflects that only interactions among interfaces are adequate in good designs. These external relationships could be generalisations (*ge*) or not directed relationships like associations (*as*), aggregations (*ag*) and compositions (*co*), as they do not allow the abstraction of candidate interfaces. Dependencies are not included because they always have a direction. Since the software architect might be interested in setting certain design preferences by demoting some relationships to others, a weight (w_x) can be assigned to the number of occurrences of each type of relationship (n_x). As an example, the modeller may want to avoid dividing into different components a parent class and its subclasses, i.e. sharing data structures, which could be more harmful to the overall cohesion than just splitting a single association between them, usually involving an operational flow. This would imply assigning a higher weight value to generalisations. Therefore, *ERP* is calculated using the expression in Eq. (2), where i and j represent each pair of components in the architectural solution.

$$ERP = \sum_{i=1}^{n} \sum_{j=i+1}^{n} (w_{as} \cdot n_{as_{ij}} + w_{ag} \cdot n_{ag_{ij}} + w_{co} \cdot n_{co_{ij}} + w_{ge} \cdot n_{ge_{ij}}) \tag{2}$$

Finally, the *groups/components ratio* (*GCR*) metric, presented in Eq. (3), is inspired by the *component packing density* (*CPD*) metric defined in [25]. *CPD* calculates the ratio between the number of constituents, e.g. operations, classes or modules, and the number of components in the overall architecture. Here, the constituents are groups of interdependent classes (*cgroups*). In a graph visualisation of the model, where classes are the nodes and its relationships, the edges, each *cgroup* is a *connected component* of this graph. Since software architects prefer a set of components with a well-defined functionality, the optimal value of *GCR* is equal to its minimum, 1, meaning that each component is comprised by a unique group of strongly interrelated classes.

$$GCR = \#cgroups/\#components \tag{3}$$

Once the three design metrics have been defined, the fitness function can be calculated as a ranking aggregation, where the best values are the lowest and, consequently, the overall fitness should be minimised. A ranking method is applied over the population and independently for each metric, resulting in three ranking values that are added, as can be seen in Eq. (4), where r returns the ranking position of a specific individual.

$$fitness_{ind} = \begin{cases} r(ICD_{ind}) + r(ERP_{ind}) + r(GCR_{ind}) & \text{if } ind \text{ is } valid \\ \#individuals \cdot \#metrics + 1 & \text{if } ind \text{ is } invalid \end{cases} \tag{4}$$

Special attention is given to invalid solutions. In such a case, a high value is assigned to the individual, i.e. a fitness value even greater than the value computed for the worst valid individual. If a valid individual would have reached the worst values in all the metrics, its ranking for each metric would be equal to the number of individuals in the population, and the aggregate value would be equal to the product of the number of metrics composing the ranking and the population size. Thus, an invalid solution always has a greater fitness than any valid individual just by adding 1 to this value.

4.4. Genetic operator

Genetic operators allow the creation of new solutions from others. Here, a mutation operator is considered for exploring design alternatives. Due to the characteristics of the problem, the execution of other kinds of operators does not seem to be

applicable, as they would probably cause the replication of classes after the combination of components from different individuals.

Five mutation procedures are proposed in order to provide a variety of new solutions, simulating those architectural transformations that software architects could manually apply during the discovery process. Domain knowledge is properly used in most cases, being an important success factor, as some of them have a great impact in the structure of the resulting architecture. Next, the description of each procedure is detailed.

Add a component. A new component is added to the architecture. Since empty components are not valid, one or more classes are selected from others to be inserted into the new one. The underlying heuristic considers the number of groups of classes inside the rest of components as a decision factor. More precisely, components built with more unconnected groups (which probably do not present a well defined functionality) are considered better candidates to provide classes than those with a unique group of classes.

At this point, the heuristic procedure uses the expression in Eq. (5) as a probability threshold of selection of each component i to act as contributor. As can be seen, this formula calculates a probabilistic value for each component i as the ratio between its number of groups of classes ($\#cgroups_i$) and the maximum number of groups ($max_{cgroups}$) corresponding to some component j of the architecture. Thus, the higher the number of groups inside the component i, the greater the probability of selecting some of its groups.

$$\begin{aligned} &Prob(i_{contributor}) = \#cgroups_i / max_{cgroups} \\ &max_{cgroups} = max(\#cgroups_j)\ j \in [1, n] \end{aligned} \tag{5}$$

The complete heuristic procedure is shown in Algorithm 1. Firstly, variables are initialised and the probability of "acting as contributor" is calculated for each component. If a random generated value surpasses the probability threshold, the groups of classes inside the component are obtained (lines 4–5). If the component comprises more than one group, their size (i.e. the number of classes composing it) is calculated and the smallest groups are searched (lines 6–13). Notice that small-sized groups are preferable because the new component could also receive groups of classes from others. Thus, a group of classes between those candidates, i.e. the smallest groups, is randomly selected, and its classes are removed and inserted into the new component, while the rest of the component is copied in the offspring (lines 14–16). The process is repeated for each available component in the parent. If no component in the parent meets the requirements (all of them presents a unique group of classes), or the randomness of the result cannot be guaranteed (only one candidate exists, so the descendant would be always the same), the new component is generated completely at random, extracting classes from all existing components (lines 18–30).

Algorithm 1. Add a component

```
Require: parent
Ensure: offspring
 1: offspring ← ø
 2: for all component in parent do
 3:   candidates ← ø
 4:   if (random(0,1) > Prob(component)) then
 5:     allGroups ← getGroups(component)
 6:     if (size(groupsOfClasses) > 1) then
 7:       for all groupOfClasses in allGroups do
 8:         if (size(groupOfClasses) = min) then
 9:         candidates ← groupOfClasses
10:          end if
11:       end for
12:       end if
13:   end if
14:   newComp ← randomGroup(candidates)
15:   offspring ← component − candidates
16:   candidates ← ø
17: end for
18: if (newComp == ø) then
19:   offspring ← ø
20:   for all component in parent do
21:     for all class in component do
22:       if (random(0,1) > 0.5) then
23:         candidates ← class
```

```
24:         end if
25:       end for
26:       offspring ← component − candidates
27:       newComp ← candidates
28:       candidates ← ø
29:     end for
30: end if
31: offspring ← newComp
32: setInterfacesAndConnectors(offspring)
33: return offspring
```

At the end, the new component is added to the offspring (line 31) and the interfaces and connectors have to be arranged (line 32), considering that the new distribution of classes may produce changes in the interactions among components. More precisely, interfaces are moved from the contributors to the new component if the classes that would implement these interfaces have been displaced. At this point, two circumstances can occur: (a) an interface remains in the new component because the interaction target continues to exist within the original component, or (b) both interfaces must be removed, since the classes specifying them have been allocated in the new component, and the interaction only happens internally. Similarly, the movement or loss of interfaces may also affect the number of connectors.

Fig. 2b corresponds to the resultant individual after the application of this mutation procedure over the individual shown in Fig. 2a. As can be seen, the movement of classes *F* from *Component_2* and *B* from *Component_1* implies that interface *B_req_D* is also removed from *Component_2* and allocated in the new one (*Component_3*). After that, *Component_1* interacts with *Component_3*, providing it some services, instead of with *Component_2*.

Split a component. One component is divided into two new components. The heuristic firstly tries to randomly select a component among those with more than one group of classes. In Algorithm 2, candidate components are identified (lines 3–7). If more than one candidate exists, one of them is randomly selected and each of its inner groups is randomly allocated in one of the two new components with equal probability (lines 8–17). If all components present a unique group of classes, one component in the parent is chosen and its classes are randomly distributed (lines 18–26). Then, all components in the parent except the component to be split are copied, and the two new components are also added (lines 27–29). Finally, interfaces and connectors are identified again, as the creation of new components can produce the appearance of new interactions (line 30).

Algorithm 2. Split a component

```
Require: parent
Ensure offspring
 1: offspring ← ø
 2: candidates ← ø
 3: for all component in parent do
 4:   if (numberOfGroups(component) > 1) then
 5:     candidates ← component
 6:   end if
 7: end for
 8: if (size(candidates) > 0) then
 9:   compToSplit ← randomComponent(candidates)
10:   for all groupOfClasses in compToSplit do
11:     if (random(0,1) > 0.5) then
12:       component1 ← groupOfClasses
13:     else
14:       component2 ← groupOfClasses
15:     end if
16:   end for
17: else
18:   compToSplit ← randomComponent(parent)
19:   for all class in compToSplit do
20:     if (random(0,1) > 0.5) then
21:       component1 ← class
22:     else
```

(continued on next page)

```
23:         component2 ← class
24:       end if
25:     end for
26: end if
27: offspring ← parent − compToSplit
28: offspring ← component1
29: offspring ← component2
30: setInterfacesAndConnectors(offspring)
31: return offspring
```

Remove a component. One component will be removed and its inner classes, randomly distributed among the remaining components. An aim of this operator is to improve the solution by reducing the ERP metric. As can be seen in Algorithm 3, the number of external relations outside the bounds of each component is obtained and those with the highest value are selected (lines 4–8). Then, a random component is chosen among them and the rest of components are copied in the offspring (lines 9–10). Next, the inner classes of the removed component are randomly distributed in the remaining components of the offspring (lines 11–13).

Algorithm 3. Remove a component

```
Require: parent
Ensure offspring
 1: offspring ← ø
 2: candidates ← ø
 3: maxRel ← maxNumExtRel(parent)
 4: for all component in parent do
 5:   if (numExtRel(component) == maxRel) then
 6:     candidates ← component
 7:   end if
 8: end for
 9: compToRemove ← randomComponent(candidates)
10: offspring ← parent − compToRemove
11: for all class in compToRemove do
12:   randomComponent(offspring) ← class
13: end for
14: setInterfacesAndConnectors(offspring)
15: return offspring
```

Finally, interfaces and connectors are checked in the offspring (line 14). In this case, interfaces from the removed component are either bound to other components when they have received the corresponding classes, i.e. those specifying the required or provided service, or removed, if the target component was the owner of the other interaction point.

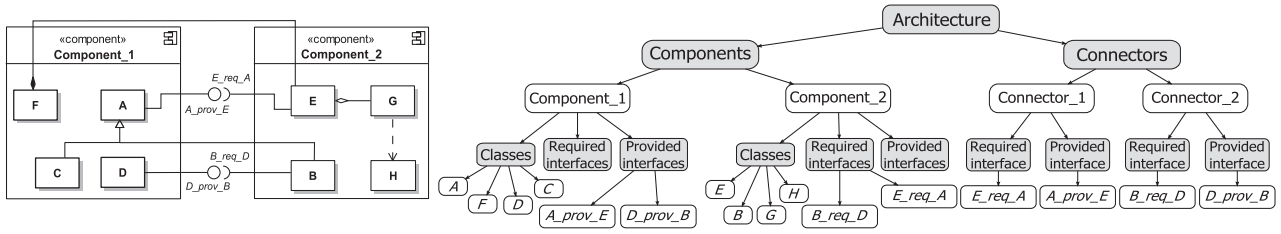
Merge two components. The elements of two previously selected components are all put together into a new component. The proposed procedure, detailed in Algorithm 4, looks for the reduction of the ERP metric. As can be seen, one of the two components taking part in the mutation is the component having the highest number of external relations (lines 4–9). When some components present the highest values, two of them are selected (lines 10–11). If not, the other component is randomly selected between the rest of components in the parent (lines 12–13). Next, components not selected in the parent are copied in the offspring, as well as the union of the two selected components (lines 15–16). Finally, interfaces and connectors must be compacted due to the merge of the two components, so the previous interactions between them are discarded (line 17).

Algorithm 4. Merge two components

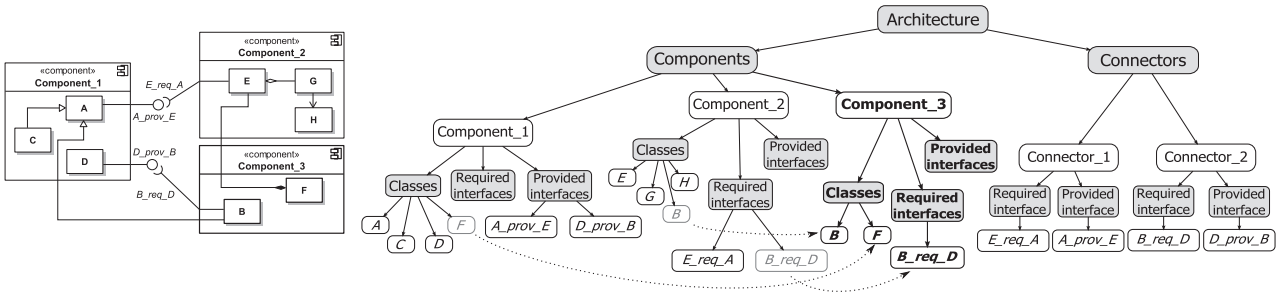
```
Require: parent
Ensure offspring
 1: offspring ← ø
 2: candidates ← ø
 3: maxRel ← maxNumExtRel(parent)
```

4: **for all** *component* in *parent* **do**
5: **if** (numExtRel(*component*) == *maxRel*) **then**
6: *candidates* ← *component*
7: **end if**
8: **end for**
9: *component*1 ← randomComponent(*candidates*)
10: **if** (size(*candidates*) > 1) **then**
11: *component*2 ← randomComponent(*candidates*)
12: **else**
13: *component*2 ← randomComponent(*parent*)
14: **end if**
15: *offspring* ← *parent* − *component*1 − *component*2
16: *offspring* ← *component*1 ∪ *component*2
17: setInterfacesAndConnectors(*offspring*)
18: **return** *offspring*

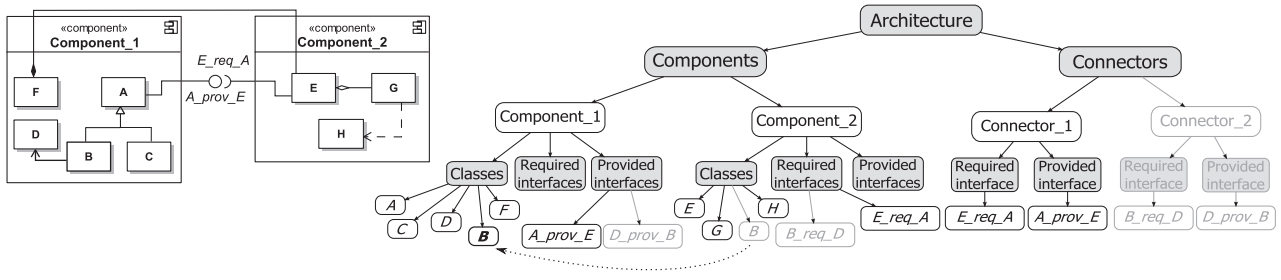
Move a class. A simple movement of a class from one component to another is performed. As it is the least destructive procedure in terms of structural modifications of the original solution, both the class and the source and target components are always randomly selected (see lines 2–4 of Algorithm 5). The selected class is removed from the origin component and added to the destination component (lines 5–6). Then, the original and modified components are copied to the offspring (lines 7–9). Since the class repositioning could also imply the creation of new interfaces in the target component and its elimination from the source, or vice versa, interfaces and connectors are revised (line 10).



(a) Initial individual



(b) Add a component mutation procedure



(c) Move a class mutation procedure

Fig. 2. Examples of mutation procedures.

Algorithm 5. Move a class

Require: *parent*
Ensure *offspring*
1: *offspring* ← ø
2: *origin* ← randomComponent(*parent*)
3: *destination* ← randomComponent(*parent*)
4: *class* ← randomClass(*origin*)
5: removeClass(*class*,*origin*)
6: addClass(*class*,*destination*)
7: *offspring* ← *parent* − (*origin* ∪ *destination*)
8: *offspring* ← *origin*
9: *offspring* ← *destination*
10: setInterfacesAndConnectors(*offspring*)
11: **return** *offspring*

An example of the application of this procedure is shown in Fig. 2c. In the original individual (see Fig. 2a), class *B* is chosen and moved from *Component_2* to *Component_1*. This operation also affects the interaction between both components, since interfaces *D_prov_B* and *B_req_D* disappear because the classes *B* and *D* belong now to the same component.

After the explanation of the different mutation procedures proposed, the mutator itself can be described. As detailed in Algorithm 6, a probabilistic roulette is built for each parent comprising only those mutation procedures that could be applied (lines 4–8). For example, a component cannot be removed if the individual already comprises the minimum number of components. Once the roulette is completed, a mutation procedure can be randomly selected according to its configured weight (line 9). If the resulting individual does not satisfy all the architectural constraints, a new mutation is performed until a valid individual is obtained or a maximum number of attempts is reached (lines 10–19).

Algorithm 6. Mutation operator

Require: *parent*, *weights*
Ensure *offspring*
1: *offspring* ← ø
2: *roulette* ← ø
3: *selectedMutator* ← ø
4: **for all** *mutator* in *mutators* **do**
5: **if** (isApplicable(*mutator*, *parent*)) **then**
6: *roulette* ← *mutator*
7: **end if**
8: **end for**
9: *selectedMutator* ← rouletteSelection(*roulette*,*weights*)
10: *attempts* ← 0
11: *invalid* ← *TRUE*
12: **while** (*invalid* == *TRUE and attempts* < 10) **do**
13: *offspring* ← mutate(*selectedMutator*, *parent*)
14: **if** (isInvalid(*offspring*)) **then**
15: *attempts* + +
16: **else**
17: *invalid* ← *FALSE*
18: **end if**
19: **end while**
20: **if** (isInvalid(*offspring*) **and**
random(0,1) < $Threshold_{invalid}$) **then**
21: *offspring* ← *parent*
22: **end if**
23: **return** *offspring*

If all attempts fail and no valid solution is found, the mutated individual could survive (lines 20–22) depending on the stage of the evolution process. A probabilistic method is proposed in order to determine whether this invalid individual will be considered as a candidate to be part of the new population, i.e. an offspring in the survival competition. A dynamic threshold, $Threshold_{invalid}$, which decreases with the elapse of generations (*gener*), is calculated in Eq. (6). Notice that, at the beginning of the algorithm, invalid individuals are permitted. Nevertheless, less invalid solutions generated by the mutator will survive due to the dynamic decrease of the threshold during the evolutionary process. Then, the replacement strategy determines whether the invalid individual will finally be part of the next generation.

$$Threshold_{invalid}(gener) = (\#generations - gener)/\#generations \tag{6}$$

4.5. Algorithm

The proposed algorithm (see Algorithm 7) follows the classical generational scheme. Firstly, some preprocessing is required (lines 1–3) in order to extract classes and its relationships from the analysis model (*classDiagram*). Then, candidate interfaces are identified using the information comprised by these relationships. Connectors are not explicitly obtained at this step, as they depend on the association of two specific candidate interfaces, and this process is performed during the creation of individuals. Next, these elements are used in combination with the number of individuals (*nInds*) and the minimum and maximum in the number of components (*minComp* and *maxComp* respectively), to initialise the population (line 4). Then, individuals are evaluated (line 5) and the iterative process begins. In each generation, parents are selected (line 8) and mutated (line 9) according to the mutation weights (*weights*). Candidate individuals must be evaluated next (line 10), so metrics are computed over them and the ranking fitness function is calculated. Note that this evaluation requires both the offsprings and the actual population in order to assign rankings in a proper way. Finally, the replacement strategy (line 11) chooses those individuals that will survive, assigning them to the next population. When the maximum number of generations (*maxGen*) is reached, the evolution ends and the best individual in the current population is returned as the candidate architecture (lines 14–15).

Algorithm 7. Proposed evolutionary algorithm

Require: *classDiagram*, *nInds*, *maxGen*, *weights*, *minComp*, *maxComp*
Ensure *candidateArchitecture*
1: *classes* ← extractClasses(*classDiagram*)
2: *relationships* ← extractRelationships(*classDiagram*)
3: *interfaces* ← identifyInterfaces(*relationships*)
4: *population* ← create(*nInds*, *minComp*, *maxComp*, *classes*, *interfaces*)
5: evaluate(*population*, *relationships*)
6: *generation* ← 0
7: **while** *generation* <= *maxGen* **do**
8: *parents* ← select(*population*)
9: *offsprings* ← mutate(*parents*, *weigths*)
10: evaluate(*population* ∪ *offsprings*, *relationships*)
11: *population* ← replace(*population* ∪ *offsprings*)
12: *generation* + +
13: **end while**
14: *candidateArchitecture* ← best(*population*)
15: **return** *candidateArchitecture*

5. Experimentation

The complete approach and all the experiments performed have been written in Java. Additionally, its functionalities have been supported by diverse publicly available Java libraries. SDMetrics Open Core[1] offers some utilities for parsing XMI files, the most commonly used XML format for model interchange, providing interoperability and serialisation across different modelling tools. Thus, the proposed approach provides support to directly collect information from analysis models created, in this case, with MagicDraw tool.[2] The Datapro4j library[3] has been used to preprocess and manage internal data structures. Finally, the evolutionary algorithm has been coded using the JCLEC framework [37].

[1] http://www.sdmetrics.com/OpenCore.html.
[2] http://www.nomagic.com/.
[3] http://www.uco.es/grupos/kdis/datapro4j.

The experiments were run on a HPC cluster of 8 compute nodes with Rocks cluster 6.1 x64 operating system. Each node comprises two Intel Xeon E5645 CPUs with 6 cores at 2.4 GHz and 24 GB DDR memory.

5.1. Problem instances

Seven system designs were used for experimentation, allowing a variety of complexity in both number of classes and number of candidate interfaces. Table 1 shows the characteristics of the problem instances considered. The interfaces column (*#Interfaces*) represents the number of relationships among classes where its navigability is explicitly specified, i.e. the number of candidate interfaces. Note that the total number of relationships (navigable in one or both directions) is also included and categorised by the types of relations defined by UML 2: associations (*As*), dependencies (*De*), aggregations (*Ag*), compositions (*Co*) and generalisations (*Ge*).

Focusing on the nature of the software models, it is worth mentioning that six of them belong to real working systems, whereas the first one, *Aqualush*[4] is a benchmark used for educational purposes. All of them apart from *Datapro4j* can be accessed from the Java Open Source Code Project Website.[5]

5.2. Parameter study

Due to the complexity of the problem, the performance of an accurate parameter study is recommended in order to analyse their suitability and influence. Firstly, different selection and replacement methods are combined and proved in order to check its influence in two important factors: the selection pressure and the capability to remove invalid solutions. Additionally, the behaviours shown by setting different weights associated to the roulette of mutation procedures (see Section 4.4) permit analysing their influence on the quality and type of returned solutions. Finally, other parameters, like the number of generations or the population size, need to be considered, since they represent key aspects in the evolutionary performance.

Regarding *RQ2*, the aim here is to obtain the most fitting setup for the proposed algorithm, whilst also providing some guidelines on the parameters that can be helpful to the software architect, who is likely not to be an expert in optimisation techniques. In this sense, we want to stress the ability of the algorithm to serve as a generic framework for architecture optimisation, where different types of solutions can be simultaneously evolved.

5.2.1. Selection and replacement strategies

Selection and replacement procedures constitute important factors in the algorithm design. Selection determines the way in which individuals are chosen to be mutated, whereas the replacement defines the type of survival competition between them. The selection methods probed are the following:

- *Deterministic selector* (DS): Each individual in the population is selected to act as a parent.
- *Tournament selector* (TS): A binary tournament is performed as often as the number of individuals in the population, in order to generate the same number of descendants than the previous method.
- *Roulette selector* (RS): A roulette is applied to select the parents. In the same way, the process is applied until the number of parents reaches the population size.

Focusing on the replacement strategies, some special constraints are considered in the replacement methods that are given below. Firstly, the best individual in the current population will survive. Secondly, when some type of competition must be established between a current invalid solution and a generated invalid descendant, both having the same maximum fitness value, the descendant is preferred, promoting the evolution of the population. The strategies considered in the preliminary study are the following:

- *Best individuals* (BR): The best n individuals from parents and descendants are selected to conform the new population, n being the population size.
- *Parent/descendant competition* (CR): A competition between each parent and its descendant is performed, and only the best survives.
- *Elitism (1) and descendants* (EL1R): After saving the best individual found in the current population, the rest of the population is filled with the $n - 1$ best descendants.
- *Elitism (10%) and descendants* (EL10R): Similar to EL1R, but keeping a major percentage of individuals from the current population.
- *Binary Tournament* (TR): all individuals are participants of the tournament, and the n best individuals are selected for the next population.

[4] http://www.ifi.uzh.ch/rerg/research/aqualush.html.
[5] http://java-source.net/.

Table 1
Problem instances and its internal properties.

Problem	#Classes	#Relationships					#Interfaces
		As	*De*	*Ag*	*Co*	*Ge*	
Aqualush	58	69	6	0	0	20	74
Borg	167	44	109	36	38	90	300
Datapro4j	59	3	4	3	2	49	12
Java2HTML	53	20	66	15	0	15	170
JSapar	46	7	33	21	9	19	80
Marvin	32	5	11	22	5	8	28
NekoHTML	47	6	17	15	18	17	46

To perform an accurate experimentation and setup, each selection method has been combined with the aforementioned replacement strategies, resulting in 15 different algorithm variants. Then, 30 executions for each algorithm version have been performed with different random seeds. The rest of the parameters are fixed to default values. All mutation procedures have the same probability to be executed, 0.2 being the corresponding weight for each one. The default minimum and maximum number of components is set to 2 and to the number of classes within the original analysis model, respectively. Here, the maximum limit has been fixed to 8, providing a wide enough range of types of solutions for the considered problem instances. The weights for the different types of UML relationships, used in the ERP metric, are internally fixed using the following configuration: $w_{as} = 2,\ w_{ag} = 3,\ w_{co} = 3,\ w_{ge} = 5$. Finally, 100 individuals and 100 generations complete the parameter configuration at this point of the study.

The first analysis of the obtained results has consisted in the evaluation of two important criteria: (a) the ability of removing invalid solutions and (b) an appropriate convergence of the population. Some variants have been discarded, as they do not achieve a final population of valid individuals, owing to an inappropriate selection pressure. This situation frequently occurs with the replacement based on TR, especially with the most difficult problem instances. On the contrary, other versions suffer an excessive convergence, so they too are rejected. In this case, the main factor that promotes this fact is the replacement strategy, since BR and CR methods strongly encourage the survival of the best individuals and also lead the search towards the same type of architectural solutions.

After this preliminary study, those variants showing an appropriate behaviour in terms of the criteria aforementioned have been analysed considering the best solutions found for all the problem instances. The Friedman test was applied to statistically validate these results, where the null hypothesis, H_0, determines that all the remaining variants perform equally well. Next, the Holm post hoc test was used when H_0 was rejected with a significance level of 95% ($\alpha = 0.05$).

Since the ranking value reached by the best individual is relative to the population to which it belongs, fitness values are not directly comparable among different executions. Therefore, the aggregate rankings of all the individuals returned by each variant and execution were recomputed. Adding the ranking obtained for every individual for a given algorithm, i.e. aggregating the results of the 30 executions per algorithm, the quality of the obtained solutions can be estimated in a proper way. The first column in Table 2 compiles the results after applying the Friedman test to these representative values.

As can be observed, *DS-EL10R* obtains the lowest ranking value. The corresponding value of the Iman and Davenport statistics, called z, is 0.4312, whereas the critical value, according to the F-Distribution with 5 and 30 degrees of freedom, the p-value, is 2.5336. Since p-value $> z$, H_0 cannot be rejected. At this point, there are no significant differences between them, and a further analysis is still required.

The preceding procedure has been repeated considering the values of each metric associated with the fitness formula separately, i.e. ICD, ERP and GCR. Table 2 shows the average ranking values after performing the Friedman test over the corresponding metrics in the best individuals found. The value of the statistics, according to the aforementioned Iman and Davenport procedure, and the conclusion about the null hypothesis are also included. Thus, significant differences exist for the ICD and ERP metrics (highlighted in bold typeface), z being greater than the p-value (2.5336).

In order to reveal those differences regarding ICD and ERP, the Holm test has been performed as a post hoc procedure. Tables 3 and 4 detail the obtained results for ICD and ERP metric, respectively. As for the ICD, the algorithm *RS-EL10R* obtained the best average ranking in the Friedman test, so it is the control algorithm. At a significance level of $\alpha = 0.05$, Holm test rejects the hypothesis that the algorithm performs equally well than the control algorithm when p-value < 0.0167. Regarding the correspondent α/i column, *RS-EL10R* is statistically better than *DS-EL10R* and *TS-EL1R*. Related to ERP, the same procedure can be realised. In this case, *TS-EL1R* acts as the control algorithm, and significant differences can be appreciated when p-value < 0.0167. As can be seen, *TS-EL1R* is better, in terms of ERP, than *RS-10R* and *TS-CR*.

After this analysis, some conclusions can be drawn. As shown, ERP and ICD constitute two conflicting metrics, whilst GCD is easily optimised by the considered algorithms, since there are no significant differences between them. The algorithm with the best average ranking when comparing by fitness, *DS-EL10R*, has not such a proper behaviour, since ICD is heavily harmed in favour of ERP and GCR. On the contrary, *TS-10R* comes up as an interesting option since it shows good performance in terms of its fitness, having the second better ranking. Moreover, it can be noted that, regarding the ICD and ERP metrics, there is no significant difference between this and the control algorithm. Usually, this variant is able to discard solutions where ERP is highly optimised. Consequently, it leads to the loss of quality in the structure of the final individuals. More specifically,

Table 2
Friedman rankings for fitness and design metrics.

Algorithm	Fitness	ICD	ERP	GCR
DS-EL10R	2.8571	4.2857	2.1429	3.0000
TS-CR	4.9286	3.4286	5.4289	5.2143
TS-EL1R	3.3571	5.4286	**1.8571**	3.1429
TS-EL10R	2.9286	3.1429	3.0000	3.2143
RS-EL1R	3.2857	3.0000	3.6429	2.7143
RS-EL10R	3.6429	**1.7143**	4.9286	3.7143
z	1.1757	4.9468	9.1546	1.8132
H_0	Accepted	Rejected	Rejected	Accepted

Table 3
Holm test results for ICD.

i	Algorithm	z	p	α/i	H_0
5	TS-EL1R	3.7143	2.0378	0.0100	Rejected
4	DS-EL10R	2.5714	0.0101	0.0125	Rejected
3	TS-CR	1.7143	0.0865	0.0167	Accepted
2	TS-EL10R	1.4286	0.1531	0.0250	Accepted
1	RS-EL1R	1.2858	0.1985	0.0500	Accepted

Table 4
Holm test results for ERP.

i	Algorithm	z	p	α/i	H_0
5	TS-CR	3.5714	3.5504	0.0100	Rejected
4	RS-10R	3.0714	0.0021	0.0125	Rejected
3	RS-EL1R	1.7857	0.0714	0.0167	Accepted
2	TS-EL10R	1.1429	0.2531	0.0250	Accepted
1	DS-EL10R	0.2858	0.7751	0.0500	Accepted

notice that low values for ICD illustrate the fact that the obtained solutions comprise large components with only a few interaction paths through interfaces, similarly to the case of *DS-EL10R*. The rest of variants of the algorithm can only obtain better ICD values by getting a fairly poor performance on the ERP metric. *TS-EL10R* achieves an appropriate trade-off between the three considered metrics, obtaining lower values for ERP and GCR without requiring a considerable decrease of ICD. Furthermore, it performs well in terms of convergence along the overall evolution. Consequently, *TS-EL10R* is the variant selected for the proposed version of the algorithm.

5.2.2. Mutation weights

One important characteristic of the proposed model lies in the existence of a roulette of mutation procedures as a way to control the diversity of solutions in the evolutionary process. These weights have a direct effect on two aspects of the generated solutions: its quality, as each procedure acts guided by their heuristics, and the diversity of types of solution, since they apply changes in their structure.

Several experiments have been performed to analyse the aforementioned characteristics of the generated solutions. The proposed roulette for mutation method selection comprises, as detailed in Section 4.4, five different procedures, each having an specific weight. Considering increments of 0.1 units, each mutation procedure could have a weight in the range [0.1,0.6], 126 being the total number of possible configurations. For each of those combinations, 30 executions have been carried, keeping the default values in the remaining setup, over all the problem instances. Afterwards, the procedure detailed in Section 5.2.1 is performed once again to reassign the ranking values of the best individuals.

Due to space limitations, only two representative instances, *Marvin* and *Datapro4j*, are shown and discussed next. Figs. 3 and 4 represent five box plots, where each one shows the distribution of the overall ranking value of the algorithm when each mutator weight is fixed at certain value in the range [0.1,0.6], whereas the others are combined in order to complete the roulette (the sum of all weights must be 1). Note that if the weight is fixed to 0.6, only one configuration can be generated for the rest of procedures (all fixed to 0.1), so a single line is drawn, representing the global ranking of this combination, i.e. the sum of the rankings of the best individual found in each of the 30 executions.

As for *Marvin*, interesting tendencies of fitness variation in most of the mutation procedures can be appreciated in Fig. 3. As the probability of the *Move a class* procedure is increased, the overall ranking of the algorithm is significantly punished. On the contrary, addition of components is beneficial. The *Merge two components* and *Split a component* procedures show an intermediate behaviour, where low or median weights seem to be more appropriate. In general, a trade-off between the

removal and the addition of components is advisable to obtain an improvement of the quality of the solutions without a loss of the population diversity.

Fig. 4 shows the box plots regarding *Dataprо4j*, a problem instance having more classes and relationships. The *Add a component* procedure presents worse performance than for *Marvin*, specially for high weight values, so its variation influences more strongly here than in simpler problems. *Remove a component* and *Merge to component* present an ascendant tendency, low weights being preferable. On the contrary, the algorithm behaviour with respect to the *Split a component* procedure is similar to the observed for *Marvin*, *Dataprо4j* having a slightly higher variation for each fixed weight. Finally, it seems that the search process can get better results for this kind of problem when the movement of classes is promoted. The algorithm performance is strongly impacted by those procedures that imply a large change in the solutions.

As can be seen, the algorithm behaviour substantially relies on the current weight configuration. In general, high values are not advisable in most procedures. The removal of components deserves special mention, where low values are highly expedient to restrict its influence on the overall performance. The default weight of the *Add a component* procedure can remain unaltered, since it contributes to keep the trade-off necessary between the different problem instances. Considering the results obtained by the procedures *Split a component* and *Merge two components*, especially when applied to small-sized instances, a proper use of these procedures requires increasing their chances in the roulette selective process. Furthermore, notice that reducing too much the weight of the procedure *Move a class* could be harmful, as it is the only procedure maintaining the current structure of the individual to be mutated and, consequently, it promotes the convergence of the algorithm. This scenario is beneficial when going through the last algorithm iterations, specially for complex instances. In short, after scrutinising the results obtained by the weight combinations satisfying all the constraints aforementioned, the proposed configuration is $w_{add} = 0.2,\ w_{remove} = 0.1,\ w_{merge} = 0.1,\ w_{split} = 0.3$ and $w_{move} = 0.3$.

5.2.3. Number of evaluations and population size

The previous experiments were focused on the algorithm needs with respect to its exploration and convergence. Furthermore that, even when the algorithm is able to properly deal with invalid individuals and diverse solutions, a further analysis of the most appropriate combination between the number of evaluations and the population size is still required.

Four different population sizes, from 50 to 200 individuals, have been set. The fitness convergence has been checked every 1200 evaluations, up to 24000 evaluations. Notice that previous experiments have reach a maximum of 10,000 evaluations, 100 individuals being evolved over 100 generations. Here, each variant of the algorithm has been iterated a different number of generations to fairly compare by the number of evaluations. These variants have been executed 30 times, as well.

Since the average fitness for the different runs is not representative, the following experimentation aims to evaluate the quality of the resulting solutions in terms of the three metrics comprising their fitness value separately. Again, the discussion is focused on the systems *Marvin* and *Dataprо4j* due to space limitations. On the one hand, Fig. 5 shows the convergence of the evolutionary process for the *Marvin* problem instance. The average values of the best individual found along the evaluations in terms of ICD, ERP and GCR are depicted for each considered population size. Remind that ICD should be maximised, whereas ERP and GCR should be minimised. As can be observed, the algorithm tends to perfom worse when the population size is set to 50 individuals, specially for the ICD metric. On the contrary, the algorithms evolving 100, 150 and 200 individuals behave similarly for the three metrics. Actually, a further analysis of the optimisation process for each individual metric in relation to the number of evaluations shows that the four algorithms remain considerably steady beyond 10000 evaluations. In fact, only some slight improvement concerning the ICD is obtained. Thus, standard values for both the population size and the number of evaluations, like those considered in previous experiments, still work properly for small problem instances.

On the other hand, Fig. 6 shows how the evolution for the *Dataprо4j* problem takes place. In contrast with the *Marvin* instance, substantial differences among the different algorithms are noticeable. Firstly, the problem complexity clearly hampers the jointly optimisation of the considered metrics. In this case, the algorithm with the population consisting of 50 individuals is prone to optimise ERP and GCR better than ICD. Just the opposite occurs when the population size has been set to 200 individuals. Then, the influence of the number of evaluations becomes important, and the value previously set to 10000 is not sufficient to achieve good enough results.

Summarising the experimental findings, the recommended population size is 150, whereas the number of maximum evaluations has to be fixed between 20000 and 24000, depending on the problem complexity. Here, the number of evaluations is set to 20400 evaluations, which corresponds with 136 generations for the selected population size. In regard to *RQ2*, this study has served to find the configuration that best enhances the performance of the evolutionary algorithm. Given this configuration, the algorithm achieves good results for all the considered design metrics and problem instances, satisfactorily keeping the trade-off between exploration and exploitation.

6. An illustrative example of the approach

After explaining the setup process of the proposed algorithm, a more detailed description about how the evolutionary search operates through the generations can be illustrative. Due to space limitations, this section focuses on a simple example, which allows both intermediate and final solutions to be shown. This example requires a small number of generations and population size, fixed to 10 and 5 respectively, whilst the other parameters remain unaltered.

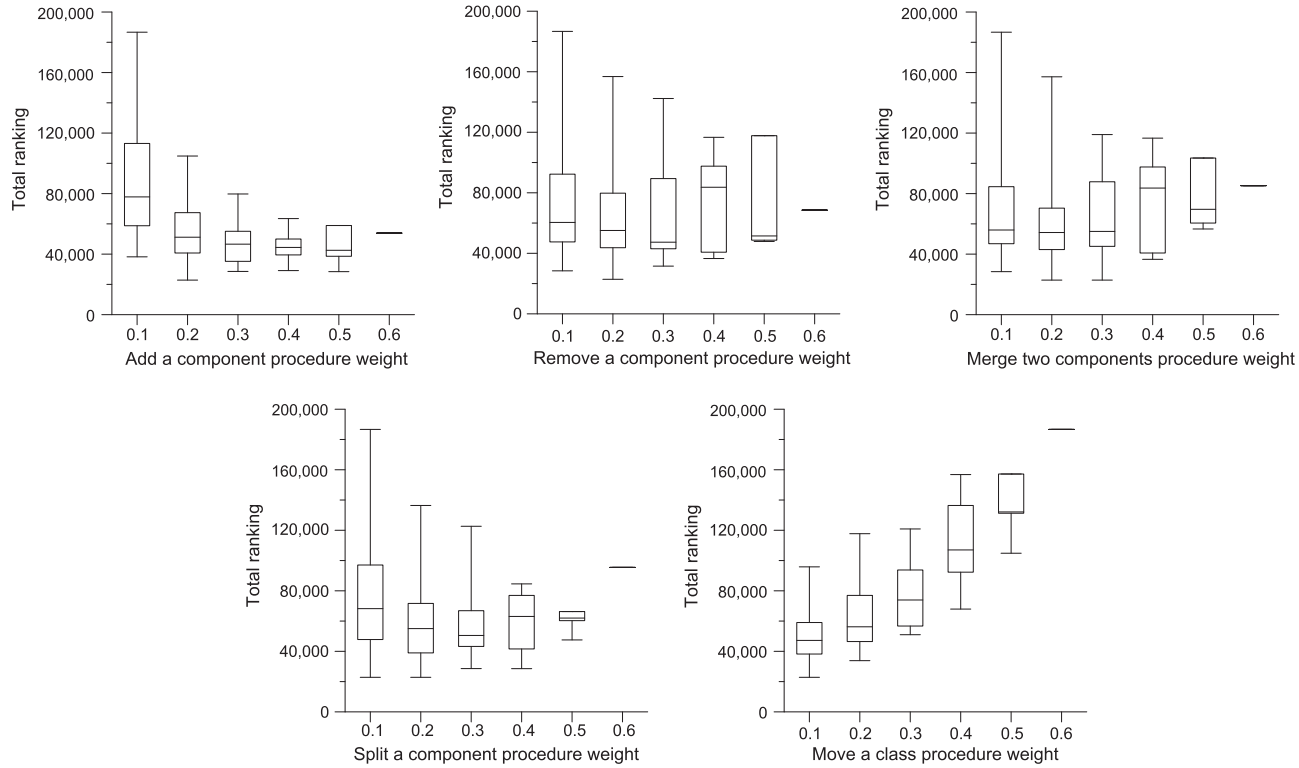


Fig. 3. Box plots of the distribution of best individuals found by the algorithm with different weights of the roulette in the mutation operator for *Marvin* problem instance.

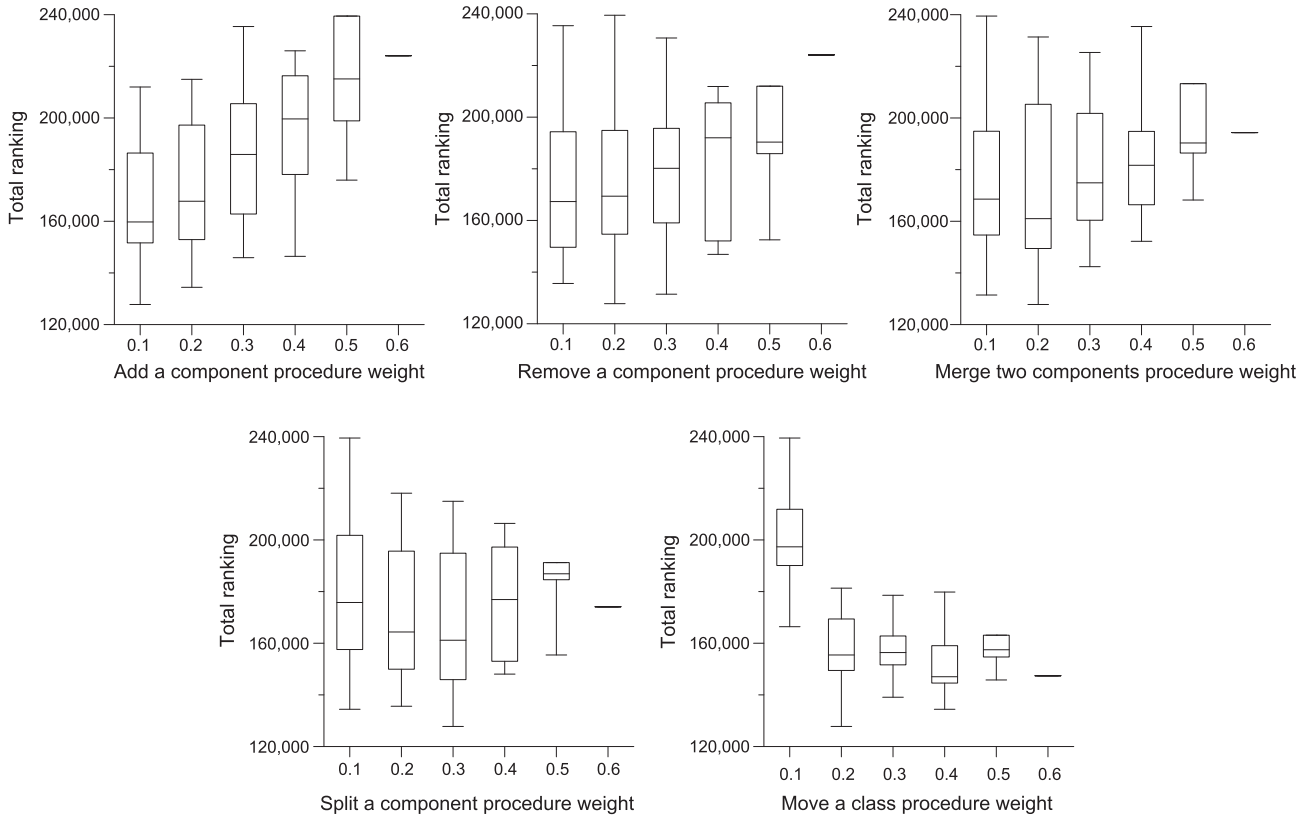


Fig. 4. Box plots of the distribution of best individuals found by the algorithm with different weights of the roulette in the mutation operator for *Datapro4j* problem instance.

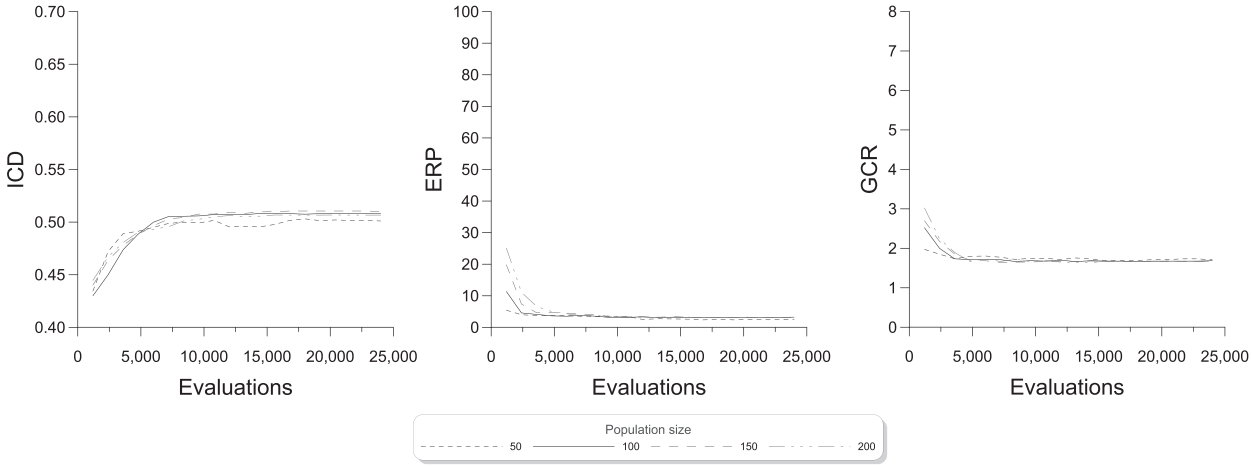


Fig. 5. Convergence of the algorithm for each selected population size (*Marvin*).

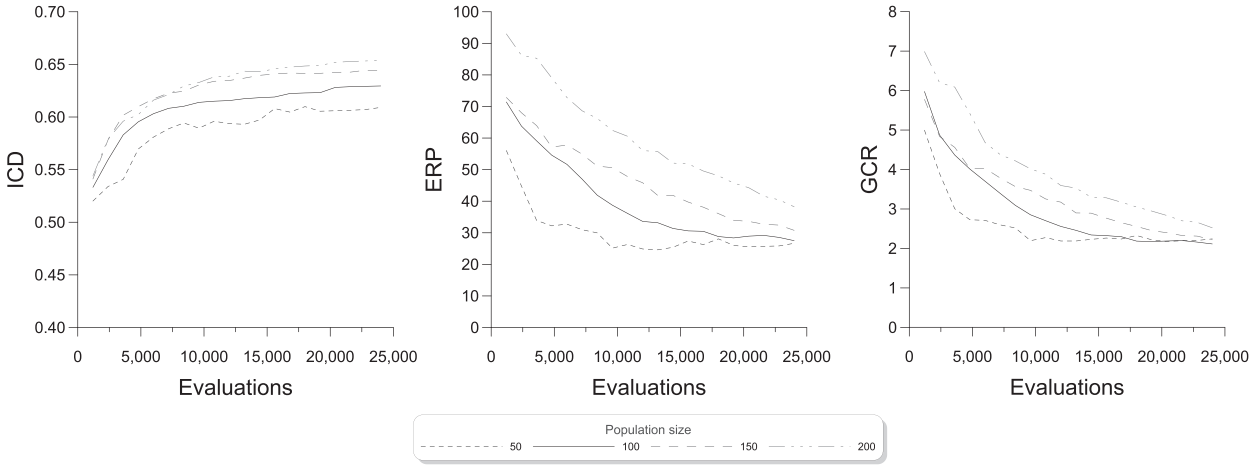


Fig. 6. Convergence of the algorithm for each selected population size (*Datapro4j*).

Fig. 7a shows the sample analysis model used as case study. As can be observed, it has 8 classes related among them with different types of relationships: 1 association, 2 dependencies, 1 composition, 1 aggregation and 2 generalisations. Moreover, two groups of well-connected classes can be distinguished: *A–B–C–D* and *E–F–G–H*. The rest of snapshots in Fig. 7 represent the phenotype of the best individuals found at different stages of the search, as well as its quality in terms of the proposed metrics and fitness values.

As can be seen in Fig. 7b, the best individual in the initial population presents an architecture composed by 3 components, in which all the classes are randomly distributed. This architecture is, as expected at this evolutionary step, a non-optimal solution. More specifically, there are some external relationships among classes belonging to different components and a few relationships among the inner classes on each one. Since two generalisations and one composition appear outside the components boundaries, then $ERP = 13.00$. Similarly, the GCR can be obtained from the number of groups (two per component) and the total number of components, so $GCR = 6.00/3.00 = 2.00$. Finally, ICD is calculated as the sum of ICD_i (from $i = 1 \ldots 3$). For this solution, its first component does not comprise any internal relationship between its inner classes and therefore $ICD = (ICD_{Component\_2} + ICD_{Component\_3})/3 = (0.21 + 0.31)/3.00 = 0.17$. At this stage, the corresponding solution has not the minimum ranking, but a low value representing that the individual is fairly good enough for some of the design criteria.

After 5 generations (see Fig. 7c), the ERP metric has been reduced because the genetic operator has grouped together more related classes. For example, classes *E*, *F*, *G* and *H* belong now to the same component, creating a well-connected group of classes that implement the functionality of *Component_2*. GCR has also been improved, since the number of groups of classes has been reduced and only *Component_2* presents more than one separate group. Finally, ICD also achieves a greater value than the previous best individual, as there is a better trade-off between internal and external interactions of components, i.e. classes are combined more adequately to provide the functionalities of the system with fewer dependencies among components. As a result of the improvements in all the values, the fitness achieves the minimum ranking value, 3.00, meaning that this individual represents the best solution in all the proposed design characteristics in comparison with the

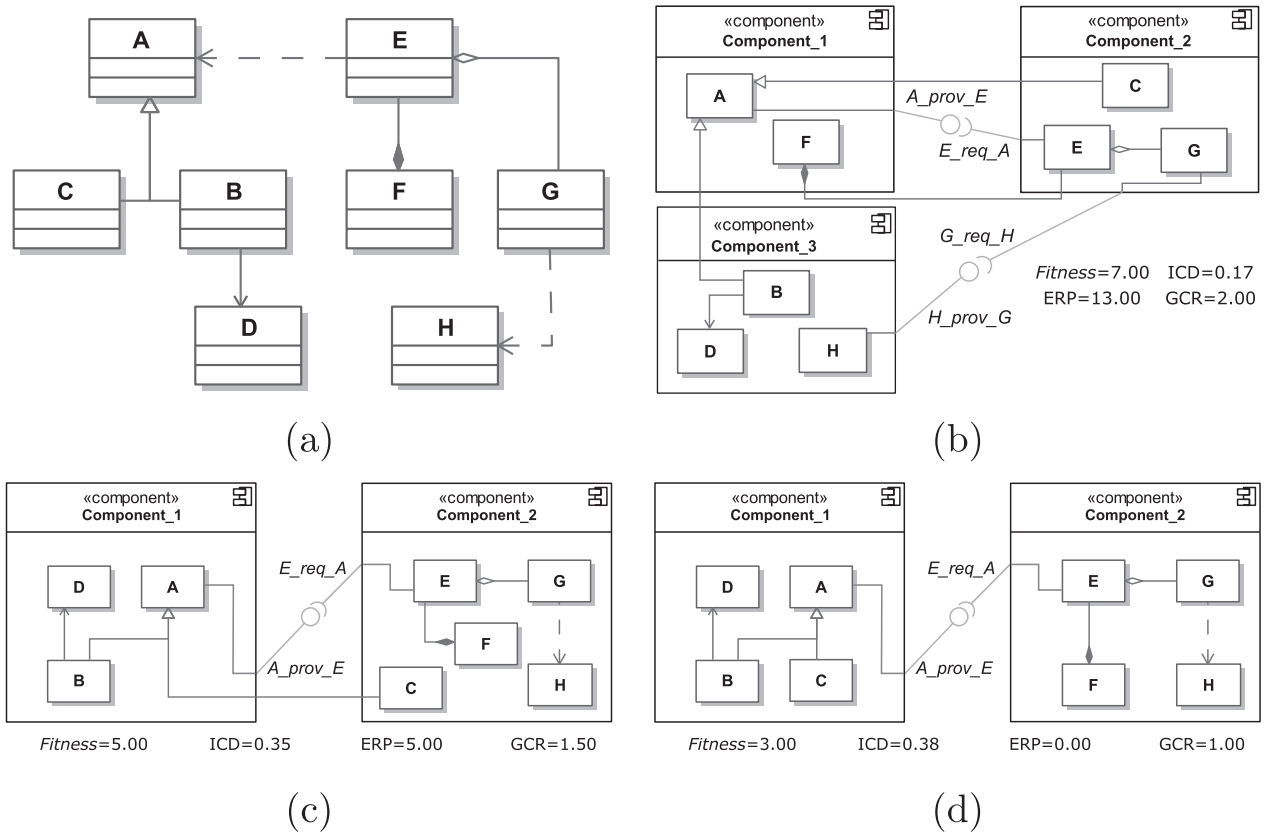


Fig. 7. Phenotype of best individual, for the class diagram which is shown in (a), found in the initial population (b), after 5 generations (c), and the final solution after 10 generations (d).

rest of members in the current population, even though it does not achieve the optimum value for every metric. For example, *ERP* = 5.00 since there is a generalisation between the separate classes *A* and *C*. Moreover, the structure of the solution has also been modified, showing that the algorithm has got to a simpler architecture.

Then, after 10 generations updating the distribution of classes (see Fig. 7d), the fitness value representing the solution quality has been significantly improved, reflecting the suitability of the solution found in terms of a good design that effectively identifies both groups of classes. Firstly, all the metrics achieve optimal values: each component presents a unique group of classes that implements its functionality, these classes are highly cohesive inside the component, and components only interact by means of a pair of interfaces. The solution also represents the simplest architecture from those obtained in previous generations, showing that the algorithm is able to adapt the structure of the solution through the evolution process.

7. Results and discussion

Table 5 shows final average results of the proposed configuration from 30 executions, including the execution time. The standard deviation is shown as subindexes. An important aspect concerning *RQ2* is that the evolutionary algorithm is able to discover good solutions [6] in the major problem instances.

For example, the algorithm achieves very good GCR results, close to the minimum, 1.0. Average fitness ranking values are omitted because they depend on each specific execution and therefore they are not representative. Nevertheless, low values are usually obtained, meaning that the architecture returned by the discovery process is the best for at least one or two of the design criteria compared to the rest of solutions found in the final population.

Focusing on the trade-off between ICD and ERP, some differences can be established between the considered problem instances. Firstly, for software designs like *JSapar* or *NekoHTML*, smaller architectures are sufficient to abstract their designs, so the ICD does not need to achieve very high values. On the contrary, with more complex problem instances (*Borg* and *Java2HTML*), the behaviour of the algorithm is quite different, where the ERP metric, which is likely to be the most difficult metric to be optimised, is highly improved. The reason is that these systems are the most complex ones, as they present a great number of interactions among classes. Therefore, the algorithm is able to find an equivalent type of solution, consisting

[6] A comparative analysis of the fitness metrics applied to a manually produced software architecture can be found at http://www.uco.es/grupos/kdis/sbse/RRV14.

Table 5
Final results of the proposed algorithm.

Problem	ICD	ERP	GCR	Time (ms)
Aqualush	$0.4124_{0.0604}$	$6.2333_{3.8443}$	$1.0833_{0.1708}$	$116.1019_{8.4891}$
Borg	$0.2820_{0.0689}$	$3.9333_{2.4626}$	$1.1667_{0.2274}$	$2489.1223_{171.2028}$
Datapro4j	$0.6425_{0.0356}$	$33.6000_{14.7436}$	$2.3989_{0.6744}$	$37.1101_{2.8308}$
Java2HTML	$0.2593_{0.0000}$	$0.0000_{0.0000}$	$1.0000_{0.0000}$	$250.4572_{7.4674}$
JSapar	$0.3751_{0.0307}$	$9.0000_{1.5492}$	$1.1667_{0.2981}$	$94.8891_{5.1418}$
Marvin	$0.5080_{0.0187}$	$3.1667_{1.0980}$	$1.6750_{0.1146}$	$14.1194_{0.6885}$
NekoHTML	$0.4594_{0.0345}$	$3.2667_{5.3037}$	$1.2389_{0.3768}$	$57.1150_{3.5524}$

in components that cover many dependent functionalities that cannot easily be dispersed in smaller components without a dramatic increase of ERP values.

Other interesting analyses can be made with *Aqualush* and *Datapro4j*, which could be considered equivalent problems, because of their similar number of classes. However, the obtained results clearly show that it is not a really relevant characteristic for the performance of the algorithm. More precisely, the identification of the *Datapro4j* architecture produces the poorest ERP value of all the considered instances. The reason is that this system is strongly hierarchical, presenting a considerable amount of generalisations in contrast to other problem instances, where associations and dependencies are the most common relationships in the system specification. This has an impact in the ERP values, since generalisations determine the maximum penalty in this measure, i.e. hierarchically-dependent classes usually tend to belong to the same component. Hence, although the number of external relationships could be similar to those obtained in the other cases, the computed ERP dramatically increases. In contrast, ICD values in *Datapro4j* are better than in *Aqualush*. This behaviour shows that the desirable trade-off between coupling and cohesion becomes difficult to achieve, not only as the amount of classes and relationships increases but also depending on the sort of software specification.

Focusing on the execution time, it is possible to determine some type of relation between the characteristics of the problem instances and the required time to perform the process. In this case, the number of relationships between classes clearly have an impact on the execution time. Medium or small instances only need a few seconds to complete the process, whereas more complex software specifications require several minutes. The underlying cause is that as the number of interactions increases, it is more difficult to generate architectures under the existing constraints, and the algorithm requires more mutation operations to obtain valid individuals.

Finally, in response to *RQ1*, some interesting information can also be extracted after studying the solutions obtained from an architectural perspective. The evolutionary algorithm provides solutions that identify and allocate well-connected groups of classes into components that correctly match with the possible intended architecture. In this way, results from the evolutionary process supply the software architect with valuable information that could be properly used to analyse the strengths and weaknesses of the system structure, reconsider some design decisions made and explore different configurations to appropriately mitigate risks. For example, some large components might be returned if the amount of relationships among classes is excessively high. Here, the software designer should remodel their interdependencies in order to get differentiate functional groups. It would reduce the system complexity and benefit maintenance and reusability. Moreover, it can be noted that the presented model is able to evolve and keep solutions with different architectural structures of interest during the search.

8. Concluding remarks

Making decisions during the software design process requires important human-centered contributions and skills that could be mitigated by search-based approaches, which are able to easily cover a great number of design alternatives. With the ultimate aim of providing support for such a decision making process, this paper presents a single-objective evolutionary approach for the discovery of component-based software architectures from analysis models, where classes and their relationships are used in the search of architectural artefacts, like components and interfaces. This proposal constitutes the first approximation to semi-automatic architectural analysis as a way to help software engineers in the improvement of their highly abstract designs which facilitate the understanding of the software foundation.

The approach is conceived as an exploratory mechanism for decision support. The underlying methodology is focused on the comprehension of the metaheuristic formulation of the problem by the software architect. Moreover, the consideration of standards like UML 2 and XMI promotes the integrability of this approach within the software engineering communities and modelling tools.

The proposed encoding is based on tree structures, similarly to the way in which specification models are handled by the different tools in this domain, bringing a flexible and intuitive representation of software architectures. Design alternatives are explored by means of five different types of mutation procedures based on those architectural transformations that software engineers usually perform for their specifications. The fitness function is calculated as the ranking aggregation of design criteria, like coupling and cohesion, as well as some specific characteristics to the problem formulation. Focusing

on the search for well-defined functionalities, the optimisation model considers the minimisation of interactions among classes and interfaces, as well as the presence of well-connected groups on classes inside the components.

The influence of the parameter setup has been discussed in detail in order to properly tune the method for semi-automatic architectural modelling, which has been tested over diverse software systems. The obtained results demonstrates the algorithm capabilities in the management of different types of solutions as well as a trade-off between the conflicting metrics, showing that the automatic discovery of the system architecture constitutes a difficult and stimulating problem.

The evolutionary approach has been conceived to deal with component-based architectures, even when it could serve as a basis for being applied to other sorts of design paradigms and areas. For example, dealing with service oriented architectures would imply a further study of the suitability of other factors, like cost and response time, whilst a model extended to comprise low-level details, like methods and properties, could serve to deal with refactoring tasks. Future research will explore the inclusion of the expert's opinion in the evolutionary search.

Acknowledgements

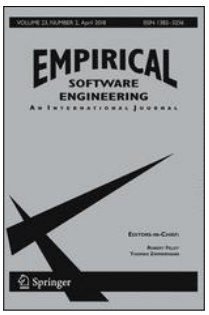
Work supported by the Spanish Ministry of Science and Technology, project TIN2011-22408, and FEDER funds. This research was also supported by the Spanish Ministry of Education under the FPU program (FPU13/01466).

References

[1] M. Abdellatief, A.B.M. Sultan, A.A.A. Ghani, M.A. Jabar, A mapping study to investigate component-based software system metrics, J. Syst. Softw. 86 (2013) 587–603.
[2] A. Aleti, B. Buhnova, L. Grunske, A. Koziolek, I. Meedeniya, Software architecture optimization methods: a systematic literature review, IEEE Trans. Softw. Eng. 39 (5) (2013) 658–683.
[3] P. Baker, M. Harman, K. Steinhofel, A. Skaliotis, Search based approaches to component selection and prioritization for the next release problem, in: 22nd IEEE Int. Conf. on Software Maintenance, 2006, pp. 176–185.
[4] M.F. Bertoa, J.M. Troya, A. Vallecillo, Measuring the usability of software components, J. Syst. Softw. 79 (3) (2006) 427–439.
[5] D. Birkmeier, S. Overhage, On component identification approaches: classification, state of the art, and comparison, in: Proc. 12th Int. Symp. on Component-Based Software Engineering, 2009, pp. 1–18.
[6] J. Bosch, P. Molin, Software architecture design: evaluation and transformation, in: Proc. IEEE Conf. and Workshop on Engineering of Computer-Based Systems, 1999, pp. 4–10.
[7] I. Boussaïd, J. Lepagnot, P. Siarry, A survey on optimization metaheuristics, Inf. Sci. 237 (0) (2013) 82–117.
[8] J.A. Clark, J.J. Dolado, M. Harman, R.M. Hierons, B.F. Jones, M. Lumkin, B.S. Mitchell, S. Mancoridis, K. Rees, M. Roper, M.J. Shepperd, Reformulating software engineering as a search problem, IEEE Proc. Softw. 150 (3) (2003) 161–175.
[9] L. Dobrica, E. Niemela, A survey on software architecture analysis methods, IEEE Trans. Softw. Eng. 28 (7) (2002) 638–653.
[10] S. Ducasse, D. Pollet, Software architecture reconstruction: a process-oriented taxonomy, IEEE Trans. Softw. Eng. 35 (4) (2009) 573–591.
[11] R. Etemaadi, M.T.M. Emmerich, M.R.V. Chaudron, Problem-specific search operators for metaheuristic software architecture design, in: Proc. 4th Int. Symp. on Search Based Software Engineering, Springer, 2012, pp. 267–272.
[12] J. Ferrer, P.M. Kruse, F. Chicano, E. Alba, Evolutionary algorithm for prioritized pairwise test data generation, in: Proc. 14th Genetic and Evolutionary Computation Conference, 2012, pp. 1213–1220.
[13] D. Garlan, Software architecture: a roadmap, in: Proc. 22th Int. Conf. of Software Engineering, 2000, pp. 91–101.
[14] P.D.S. Gui Gui, Measuring software component reusability by coupling and cohesion metrics, J. Comput. 4 (9) (2009) 797–805.
[15] P. Gupta, S. Verma, M. Mehlawat, Optimization model of COTS selection based on cohesion and coupling for modular software systems under multiple applications environment, in: Computational Science and Its Applications, LNCS, vol. 7335, Springer, 2012, pp. 87–102.
[16] M. Harman, Software engineering meets evolutionary computation, Computer 44 (10) (2011) 31–39.
[17] M. Harman, S.A. Mansouri, Y. Zhang, Search-based software engineering: trends, techniques and applications, ACM Comput. Surv. 45 (1) (2012) 11:1–11:61.
[18] S.M.H. Hasheminejad, S. Jalili, An evolutionary approach to identify logical components, J. Syst. Softw. 96 (0) (2014) 24–50.
[19] ISO. ISO/IEC FDIS 42010/D9, Systems and Software Engineering – Architecture Description, March 2011.
[20] A.C. Jensen, B.H. Cheng, On the use of genetic programming for automated refactoring and the introduction of design patterns, in: Proc. 12th Genetic and Evolutionary Computation Conference, 2010, pp. 1341–1348.
[21] S. Kebir, A.-D. Seriai, A. Chaoui, S. Chardigny, Comparing and combining genetic and clustering algorithms for software component identification from object-oriented code, in: Proc. 5th Int. C* Conference on Computer Science and Software Engineering, 2012, pp. 1–8.
[22] C. Le Goues, W. Weimer, S. Forrest, Representations and operators for improving evolutionary software repair, in: Proc. 14th Ann. Conf. on Genetic and Evolutionary Computation, 2012, pp. 959–966.
[23] S. Malek, N. Medvidovic, M. Mikic-Rakic, An extensible framework for improving a distributed software system's deployment architecture, IEEE Trans. Softw. Eng. 38 (1) (2012) 73–100.
[24] A. Martens, D. Ardagna, H. Koziolek, R. Mirandola, R. Reussner. A hybrid approach for multi-attribute QoS optimisation in component based software systems, in: Proc. 6th Int. Conf. on the Quality of Software Architectures, 2010, pp. 84–101.
[25] V.L. Narasimhan, B. Hendradjaya, Some theoretical considerations for a suite of metrics for the integration of software components, Inf. Sci. 177 (3) (2007) 844–864.
[26] OMG. Unified Modeling Language 2.4 Superstructure Specification, November 2010.
[27] C.M. Poskitt, S. Poulding. Using contracts to guide the search-based verification of concurrent programs, in: Proc. 5th Int. Symp. on Search Based Software Engineering, 2013, pp. 263–268.
[28] K. Praditwong, M. Harman, X. Yao, Software module clustering as a multi-objective search problem, IEEE Trans. Softw. Eng. 37 (2) (2010) 264–282.
[29] O. Räihä, A survey on search-based software design, Comput. Sci. Rev. 4 (4) (2010) 203–249.
[30] P. Rodríguez-Mier, M. Mucientes, M. Lama, M. Couto, Composition of web services through genetic programming, Evol. Intell. 3 (2010) 171–186.
[31] O. Sievi-Korte, E. Mäkinen, T. Poranen, Simulated annealing for aiding genetic algorithm in software architecture synthesis, Acta Cybernetica 21 (2) (2013) 235–265.
[32] C.L. Simons, I.C. Parmee, Elegant object-oriented software design via interactive, evolutionary computation, IEEE Trans. Syst., Man, Cybern. C, Appl. Rev. 42 (6) (2012) 1797–1805.
[33] C.L. Simons, I.C. Parmee, R. Gwynllyw, Interactive, evolutionary search in upstream object-oriented class design, IEEE Trans. Softw. Eng. 36 (6) (2010) 798–816.
[34] J. Smith, D. Stotts, SPQR: flexible automated design pattern extraction from source code, in: Proc. 18th IEEE Int. Conf. on Automated Software Engineering, 2003, pp. 215–224.

[35] C. Szyperski, Component Software: Beyond Object-Oriented Programming, second ed., Addison-Wesley Longman, Boston, MA, 2002.
[36] L. Troiano, C. Birtolo, Genetic algorithms supporting generative design of user interfaces: examples, Inf. Sci. 259 (0) (2014) 433–451.
[37] S. Ventura, C. Romero, A. Zafra, J.A. Delgado, C. Hervás, JCLEC: a java framework for evolutionary computation, Soft Comput. 12 (4) (2008) 381–392.
[38] D. Whitley, An overview of evolutionary algorithms: practical issues and common pitfalls, Inf. Softw. Technol. 43 (14) (2001) 817–831.
[39] Y. Zhang, M. Harman, S.L. Lim, Empirical evaluation of search based requirements interaction management, Inf. Softw. Technol. 55 (1) (2013) 126–152.

6.2. A comparative study of many-objective evolutionary algorithms for the discovery of software architectures



Title	A comparative study of many-objective evolutionary algorithms for the discovery of software architectures
Authors	A. Ramírez, J.R. Romero, S. Ventura
Journal	Empirical Software Engineering
Volume	21(6)
Pages	2546-2600
Year	2016
Editorial	Springer
DOI	10.1007/s10664-015-9399-z

IF (JCR 2016)	3.275
Category	Computer Science, Software Engineering
Position	7/106 (Q1)
Cites	2 (WoS), 4 (Scopus)

Empir Software Eng (2016) 21:2546–2600
DOI 10.1007/s10664-015-9399-z

A comparative study of many-objective evolutionary algorithms for the discovery of software architectures

Aurora Ramírez[1] · **José Raúl Romero**[1] · **Sebastián Ventura**[1]

Published online: 28 September 2015

Abstract During the design of complex systems, software architects have to deal with a tangle of abstract artefacts, measures and ideas to discover the most fitting underlying architecture. A common way to structure such complex systems is in terms of their interacting software components, whose composition and connections need to be properly adjusted. Along with the expected functionality, non-functional requirements are key at this stage to guide the many design alternatives to be evaluated by software architects. The appearance of Search Based Software Engineering (SBSE) brings an approach that supports the software engineer along the design process. Evolutionary algorithms can be applied to deal with the abstract and highly combinatorial optimisation problem of architecture discovery from a multiple objective perspective. The definition and resolution of many-objective optimisation problems is currently becoming an emerging challenge in SBSE, where the application of sophisticated techniques within the evolutionary computation field needs to be considered. In this paper, diverse non-functional requirements are selected to guide the evolutionary search, leading to the definition of several optimisation problems with up to 9 metrics concerning the architectural maintainability. An empirical study of the behaviour of 8 multi- and many-objective evolutionary algorithms is presented, where the quality and type of the returned solutions are analysed and discussed from the perspective of both the evolutionary

Communicated by: Marouane Kessentini and Guenther Ruhe

✉ José Raúl Romero
jrromero@uco.es

Aurora Ramírez
aramirez@uco.es

Sebastián Ventura
sventura@uco.es

[1] Department of Computer Science and Numerical Analysis, University of Córdoba, Córdoba, 14071, Spain

performance and those aspects of interest to the expert. Results show how some many-objective evolutionary algorithms provide useful mechanisms to effectively explore design alternatives on highly dimensional objective spaces.

Keywords Software architecture discovery · Search based software engineering · Many-objective evolutionary algorithms · Multi-objective evolutionary algorithms

1 Introduction

Software architects face some of the most difficult decisions in the earliest stages of project execution, since they must guarantee that their designs satisfy multiple quality criteria. At this point, only a few concrete items of information are known about the final system, even though its structure already has to be determined (Bosch and Molin 1999). In this scenario, the discovery of software architectures constitutes an abstract and complex task, where software engineers have to identify high-level architectural artefacts, like components and interfaces, from a prior analysis model. As a result, an architectural specification is obtained comprising the underlying system structure, which will serve throughout the software development to add later functionality, to get suitable designs or to ensure proper maintenance (Ducasse and Pollet 2009).

Along the design phase, software architects have to deal with multiple design alternatives that need to be factually evaluated in order to find those solutions that best meet the system requirements. However, characteristics like modularity and reusability clearly hamper the decision making process because of the lack of quantifiable measures that really would represent the quality of the resulting software designs. Therefore, efforts have been directed towards the definition of specific metrics at the architectural level (Narasimhan and Hendradjaya 2007; Sant'Anna et al. 2007) like those already existing and commonly accepted for more concrete artefacts (Bansiya and Davis 2002). Nevertheless, selecting the most suitable subset of metrics is a complex task usually accomplished by experts only guided by their own experience. Since many different design criteria have to be simultaneously balanced, they may affect the manner in which architectural constraints are considered to meet the software architects expectations.

The automatic evaluation of design alternatives constitutes an interesting application field for Evolutionary Computation (EC) according to the rationale used by Search Based Software Engineering (SBSE) (Harman et al. 2012), where traditional Software Engineering (SE) activities, usually performed by humans, are formulated as search problems to be solved using metaheuristics. In the last years, EC has been successfully applied to the resolution of combinatorial problems in Software Engineering like requirements selection (Durillo et al. 2011) or resource allocation in project scheduling (Luna et al. 2014). In this sense, other SE tasks have demanded the attention of more sophisticated EC approaches in order to provide a higher performance in terms of the quality of the solutions and the satisfaction of the engineer's expectations when more than one or two objectives should be considered (Yao 2013).

In this context, a trade-off between different objectives is required, resulting in a set of equivalent solutions among which the expert will decide. Well-known multi-objective evolutionary algorithms (MOEAs) (Zhou et al. 2011), like SPEA2 (Strength Pareto Evolutionary Algorithm 2) and NSGA-II (Nondominated Sorting Genetic Algorithm II), usually arise as the first option when SE practitioners want to adopt a multi-objective approach

(Sayyad and Ammar 2013). Less often, comparative studies of different MOEAs are presented with the aim of providing a further analysis of the abilities of each proposal. These studies are really useful to gain empirical evidence of the adequacy of a specific algorithm to a given problem domain (Luna et al. 2014; Zhang et al. 2013).

These studies have traditionally considered optimisation problems with 2 or 3 predefined objectives, though there exist many SE tasks whose nature clearly demands the definition of a large number of objectives. Therefore, many-objective evolutionary algorithms appear to be an interesting approach for dealing with this kind of situation (Purshouse and Fleming 2007; von Lücken et al. 2014). The current interest within the EC community in tackling highly dimensional objective spaces creates an opportunity to benefit from novel and powerful algorithms to support software design. Traditionally, many-objective approaches have been tested on benchmarks considering a continuous search space (Bader and Zitzler 2011; Yang et al. 2013; Zhang and Li 2007). For this specific kind of problems, many-objective evolutionary algorithms have shown to clearly outperform MOEAs in terms of the standard performance metrics defined within the field of multi-objective optimisation (Coello Coello et al. 2007; Zitzler et al. 2004). Only a few authors have more deeply studied the performance of many-objective approaches in real-world environments like those appearing in SE and, consequently, SBSE practitioners seem to be less familiar with this kind of algorithms. In this sense, the works presented in Kalboussi et al. (2013), Mkaouer et al. (2014), and Sayyad et al. (2013) are the first proposals that are actually focused on the application of a specific many-objective evolutionary algorithm in testing, refactoring and software product lines design activities, respectively.

The greater the number of factors required to reach architectural decisions is, the more dependent the performance, in terms of quality and feasibility, of the obtained architectural solution becomes on the selection of the most appropriate optimisation approach. Therefore, even though many-objective approaches have proven capable of providing improvements in other related fields of application, the intrinsic characteristics of each algorithm make it necessary to thoroughly study them with the aim of identifying those aspects having a greater influence on the satisfactory resolution of a given design problem. In this context, an empirical analysis allows offering a more in-depth comprehensive overview of the behaviour of different types of algorithms when dealing with a different number and sort of objectives.

In Ramírez et al. (2014), we presented an initial discussion on the performance of 5 multi- and many-objective evolutionary algorithms. This research, performed on a set of 6 different design metrics using 6 representative case studies, highlighted the suitability of this new branch of algorithms in terms of their evolutionary performance and ability to support the decision making process as the number of objectives increased. Nevertheless, we consider it necessary to broaden and deepen in different ways this previous analysis in order to get more valuable findings on the performance of different families of many-objectives algorithms applied to the automatic discovery of software architectures. Firstly, the number of problem instances has been increased to 10 real-world system specifications. Secondly, the study focuses on the maintainability of software systems, considering up to 9 metrics to represent the optimisation objectives. These design metrics quantify diverse related non-functional properties of an architectural specification. The experimental framework, which includes a detailed discussion on the existing dependencies between objectives, provides a complete study of the most suitable characteristics of the different families of algorithms to deal with the evolutionary discovery of software architectures. Amongst these families, 8 different algorithms have been chosen to carry out this empirical research. Due to the very wide range of algorithms within the multi-objective optimisation field, such an analysis has

served to articulate the distinctive features of the families of algorithms that best fit into the problem domain under study.

The rest of the paper is structured as follows. Section 2 presents some background, focused on multi- and many-objective optimisation and SBSE. Section 3 introduces the conceptual framework related to software architecture design and the search problem being addressed. The evolutionary model is explained in Section 4, including the definition of the evaluation objectives. The experimental framework is described in Section 5. Section 6 presents the obtained outcomes, whereas a discussion from the perspective of the decision making process is given in Section 7. The threats to the validity are explained in Section 8. Finally, some concluding remarks are pointed out in Section 9.

2 Background

The key concepts of multi- and many-objective optimisation are introduced in this section, as well as the evolutionary algorithms used for this comparative study. Additionally, the prior application of these multiple objective approaches in the SBSE field is reviewed.

2.1 Multi- and Many-objective Evolutionary Optimisation

Solving a multi-objective optimisation problem (MOP) consists in looking for solutions characterised by a set of decision variables that can find the most optimal trade-off among a set of objective functions, which are usually in conflict with each other (Coello Coello et al. 2007). One initial approximation to address a MOP is to turn it into a single-objective problem by formulating a new objective function that calculates the weighted sum of all the objectives. Similarly, other approaches have proposed its resolution by keeping only one objective and considering the rest of them as constraints. Even though they are just simple proposals, they have handicaps like the weight selection or requiring the reformulation of the problem (Deb 2001).

Therefore, any search algorithm specifically devoted to address MOPs will deal with objectives independently, so it can return a set of optimal solutions. Each solution is determined by a different trade-off between all the objectives, in which the improvement achieved by one could imply the loss of others. This idea is formally stated by the Pareto dominance definition (Coello Coello et al. 2007) as follows: a solution a is said to dominate another solution b if a has equal or better objective values for all the objectives and a better value for at least one objective than b. If that condition cannot be satisfied by either a or b, then both solutions are referred as non-dominated, meaning that they are incomparable or equivalent. The set of non-dominated solutions constitutes the Pareto set (PS), whereas the mapping of these solutions to the objective space is called Pareto front (PF). Finding a fitting approximation to the PF is a twofold task (Deb 2001). On the one hand, a multi-objective approach is requested to find non-dominated solutions that are close to the front, which is known as the convergence to the true PF. On the other hand, the resulting PF should be properly distributed within the objective space, making a good spread of non-dominant solutions important. Having both ideas in mind, the search algorithm should be able to provide an interesting set of high-quality solutions for the decision maker to choose among.

Evolutionary algorithms (EAs) are a metaheuristic technique inspired on the principles of natural evolution that deals with a set of solutions to solve optimisation problems (Boussaïd et al. 2013). Each solution is characterised by its genotype, i.e. the structure used to encode the decision variables that need to be managed during the problem resolution.

In contrast, the phenotype symbolises the real-world representation of its genotype. The algorithm begins with the random creation of a set of candidate solutions, known as population. Each solution, also called individual, is then evaluated considering how well it can solve the optimisation problem according to its fitness function. The rest of the evolution is performed by an iterative process in such a way that individuals in the population are modified in each iteration, also called generation, using genetic operators until the stopping condition, e.g. a maximum number of generations, is reached. With this purpose, some individuals are selected to act as parents, from which new solutions are generated according to the specific EC paradigm. The most commonly used genetic operators are crossover and mutation. The former is applied to create offspring tending to be similar to their parents, whereas the latter is performed to introduce diversity among offspring. New solutions have to be evaluated and compared against one another in order to decide whether they should be part of the next population of individuals. Finally, the best individual found in terms of its fitness function is returned. Notice that several paradigms have emerged from the general concept of Evolutionary Computation, such as genetic algorithms (GAs), evolution strategy (ES), evolutionary programming (EP) and genetic programming (GP) (Boussaïd et al. 2013). On the one hand, GA and GP apply both crossover and mutation. In the latter, individuals represent executable programs encoded using a tree structure instead of a fixed-length genotype as proposed by GAs. On the other hand, ES and EP do not usually implement any crossover operator, EP being characterised by the use of domain-specific encodings.

Algorithm 1 Pseudocode of a multi-objective evolutionary algorithm

Require: $maxGenerations$, $populationSize$
Ensure: $paretoSet$
1: $population \leftarrow$ initialisePopulation($populationSize$)
2: evaluate($population$)
3: $archive \leftarrow$ initialiseArchive($population$)
4: $generation \leftarrow 0$
5: **while** $generation <= maxGenerations$ **do**
6: $parents \leftarrow$ matingSelection($population \cup archive$)
7: $offspring \leftarrow$ variation($parents$)
8: evaluate($offspring$)
9: $population \leftarrow$ environmentalSelection($population \cup offspring \cup archive$)
10: $archive \leftarrow$ updateArchive($offspring \cup archive$)
11: $generation++$
12: **end while**
13: $paretoSet \leftarrow$ extractNonDominatedSolutions($population$, $archive$)
14: **return** $paretoSet$

Evolutionary computation has traditionally been a way to efficiently solve complex search problems having a unique fitness function. Nevertheless, they can be also adapted to cover a multidimensional objective space. In fact, the application of EAs for the resolution of multi-objective problems, i.e. multi-objective evolutionary algorithms (Coello Coello et al. 2007), has been widely studied for the resolution of real-world applications. They work well in objective spaces with 2 or 3 objectives, in both combinatorial and continuous optimisation problems. MOEAs perform in general the same iterative process than EAs. However, a specific terminology is often used Zitzler et al. (2004), and they require some additional steps to be executed, as shown in Algorithm 1. Observe that MOEAs provide a mating selection method to pick individuals to act as parents. After the generation of offspring through a domain-specific variation mechanism, i.e. genetic operators, environmental selection is performed to select the individuals that will survive in the next generation. The former can determine an overall fitness value, which is independent from

the objective values and represents a quality criterion to classify individuals, usually promoting the selection of non-dominated solutions. The latter applies some diversity preservation techniques in order to choose between equivalent solutions. In addition, an external population, called archive, can be maintained throughout the evolution containing the best found solutions. Finally, the Pareto set is returned by extracting non-dominated solutionsl from either the population or the archive. Two well-known examples of MOEAs adopting this latter structure are SPEA2 (Zitzler et al. 2001) and NSGA-II (Deb et al. 2002).

In the last decades, important efforts have been devoted to deal with the increasing complexity of MOPs, especially with those having a large number of objectives, known as many-objective optimisation problems. Although the actual difference between multi- and many-objective problems has not been clearly stated in the literature (Purshouse and Fleming 2007), some works initially considered that having more than 2 objectives can be considered as a many-objective problem (Adra and Fleming 2011). Nevertheless, most authors agree today with the idea that many-objective problems require the presence of at least 4 objectives (Deb and Jain 2014; von Lücken et al. 2014; Zhou et al. 2011). In this paper, this latter criterion is followed in order to distinguish among optimisation problems. Even so, notice that algorithms originally conceived as multi-objective have served to address many-objective problems, whereas other authors use the term multi-objective to refer to the algorithm, and the term many-objective to the optimisation problem. Here, algorithms will be categorised as prescribed by their respective authors. There are studies which conclude that proposals like SPEA2 and NSGA-II tend to decrease their performance as the number of objectives increases, at least when solving continuous optimisation benchmarks (Khare et al. 2003; Praditwong and Yao 2007). In highly dimensional objective spaces, the Pareto dominance loses the efficiency required to guide the search, since a great number of population members become non-dominated solutions. For this reason, many-objective optimisation problems require new techniques to deal with such complexity (Schutze et al. 2011; von Lücken et al. 2014).

Recently, different strategies have been applied in order to improve the performance of EAs for solving many-objective problems, including the adaptation of some already existing algorithms (Adra and Fleming 2011; Wang et al. 2014) or the design of new frameworks (Hadka and Reed 2013). Some of the proposed features are related to the modification of the dominance principle (He et al. 2014) or the inclusion of specific diversity preservation techniques (Wang et al. 2014), among others. Thus, they are usually classified in families in accordance with the selected technique (Wagner et al. 2007).

2.2 Selected Algorithms

Eight evolutionary algorithms have been selected for this comparative study. Two multi-objective evolutionary algorithms, SPEA2 and NSGA-II, are included as baseline algorithms, their original implementation being maintained. Additionally, other representative proposals, especially well-suited to deal with many-objective problems, were chosen from the different families of algorithms in accordance to the literature (von Lücken et al. 2014; Wagner et al. 2007). Notice that for this selection, algorithms could not consider any previous assumption about domain-specific elements like the encoding or the genetic operators, since both characteristics need to be configured according to the problem domain under study.

SPEA2 (Zitzler et al. 2001) assigns a fitness value based on a strength rate and a density estimator strategy. The former counts the number of dominated and non-dominated solutions considering each individual, whereas the latter uses a clustering approach to select

more diverse individuals. Both values are combined and used in the mating selection. Non-dominated solutions are stored in a fixed-size archive using a truncation method if it is required, while the new population will be set using all the offspring.

NSGA-II (Deb et al. 2002) involves a sorting method in order to rank the solutions in fronts considering the dominance between them. Both this ranking and a crowding distance are used in the mating selection. During the environmental selection, individuals are progressively kept by fronts, the crowding distance being used to decide which individuals are selected when a front cannot be stored entirely. This algorithm does not define an external archive, so the non-dominated solutions are extracted from the final population.

MOEA/D (Multiobjective Evolutionary Algorithm based on Decomposition) (Zhang and Li 2007) proposes a decomposition approach where the multi-objective problem is divided into several scalar problems to be simultaneously optimised, looking for a better diversity among solutions. More specifically, MOEA/D associates each subproblem to one individual in the population by means of a weighted vector. Individuals are randomly chosen in the mating selection, whereas the neighbourhood information is determined on the weighted vectors during the environmental selection in order to match each offspring to the subproblem that it can best solve. The Tchebycheff approach is applied to evaluate individuals over their specific problem, computing its distance to a reference point. This algorithm uses an archive to keep all the non-dominated solutions found throughout the evolution process. Originally proposed to solve MOPs, MOEA/D is frequently used in many-objective problems, as well as to try out new proposals (Deb and Jain 2014; von Lücken et al. 2014; Yang et al. 2013).

A common technique to avoid the poor selection pressure of the Pareto dominance criterion is to divide the objective space into hypercubes or grids. Then, a relaxed dominance relation, usually referred as the ϵ-dominance, is defined over the resulting landscape. Thus, solution a ϵ-dominates solution b if a belongs to better or equal hypercubes for all the objectives and to a better hypercube for at least one objective than b. This idea is explored in ϵ-MOEA (Deb et al. 2003), a steady-state algorithm which establishes a fixed length of the hypercubes for every objective, named ϵ_i. In each iteration, one parent from the current population and another one from an archive of solutions are selected to generate one or more offspring. These new solutions will survive depending on the hypercubes to which they belong and those already filled with the solutions previously saved in the external population. Similarly to MOEA/D, ϵ-MOEA was originally conceived to deal with 2 or 3 objectives, although it has turned into a reference algorithm due to its notable performance to address many-objective optimisation problems (Wagner et al. 2007; Yang et al. 2013).

GrEA (Grid-based Evolutionary Algorithm) (Yang et al. 2013) is a many-objective evolutionary algorithm based on the notion of landscape partition, though grids are here dynamically created considering the objective values in the current population. Initially, GrEA calculates some properties of the individuals based on the grids. Such information is used by the mating selection in order to obtain the most promising reproducible solutions. Inspired by NSGA-II, GrEA also ranks the population by fronts during the environmental selection, but it uses its own specific distance and diversity metrics to discard equivalent solutions. GrEA does not define an external archive, the returned solutions being extracted from the final population.

Indicator-based methods propose the use of indicators to guide the evolution process. The selected indicator can represent the preferences of the decision maker, so the evolution is heavily oriented towards a region of interest. Some popular indicator of the quality of a Pareto set approximation, e.g. the hypervolume, can be considered. This kind of algo-

rithms has become an interesting alternative to address many-objective optimisation, since they transform the original problem into a task consisting in optimising the quality indicator (von Lücken et al. 2014). One representative approach here is IBEA (Indicator-based Evolutionary Algorithm) (Zitzler and Künzli 2004), which defines a general evolutionary algorithm where any binary indicator could be used as a fitness function. The selected measure is a key factor, since the fitness function is heavily involved in both the mating and the environmental selection procedures. For instance, the authors proposed the application of the ϵ-indicator, representing the minimum distance by which a set of solutions should be translated in each dimension such that another set of solutions is weakly-dominated. It requires a less complex operation than hypervolume, which could be prohibitive in terms of computational effort when a large number of objectives is set. During the mating selection, binary tournaments are performed to generate the set of parents. For each tournament, two randomly selected individuals are compared with each other according to their fitness values, so the best individual is returned as parent. The worst individuals are discarded during the environmental selection.

Another relevant proposal is HypE (Hypervolume Estimation Algorithm) (Bader and Zitzler 2011), which allows the optimisation of many-objective problems replacing the exact value of the hypervolume with an estimated value based on Monte Carlo simulations. This value represents the portion of the hypervolume to be actually attributed to each solution and used to determine the winner of each binary tournament in the mating selection process. Environmental selection is based on the minimum loss of hypervolume, discarding those solutions that contribute the least to the overall hypervolume as a way to promote convergence to the most promising solutions in terms of this indicator. Neither IBEA nor HyPE makes use of an archive of non-dominated solutions.

Finally, NSGA-III (Deb and Jain 2014) is a many-objective evolutionary algorithm which looks for the improvement of the performance of its predecessor, NSGA-II, when dealing with highly dimensional objective spaces. It is classified as a reference-point-based method, being characterised by the promotion of those individuals that are close to a set of reference points supplied by the user or uniformly generated by the algorithm. In NSGA-III, the crowding distance of NSGA-II is replaced by a new method, where each individual is associated to the closest reference point. In this way, diversity is preserved attending to the survival of, at least, one representative solution for each reference point. Apart from belonging to a different family of many-objective approaches, NSGA-III has shown promising results when coping with complex real-world combinatorial SBSE problems like software refactoring (Mkaouer et al. 2014).

2.3 Multiple Objective Approaches in SBSE

Multi-objective optimisation algorithms have been widely applied in SBSE (Yao 2013), since they have demonstrated their ability to reach a trade-off between objectives, as is often the case in Software Engineering. Focusing on search-based software design (Räihä 2010), i.e. the field within SBSE that proposes the application of metaheuristics to the design and improvement of different software artefacts, some classical MOEAs have already been used. Object-oriented analysis using SPEA2 (Bowman et al. 2010), software module clustering executing the Two-Archive algorithm (Praditwong et al. 2011), or the application of NSGA-II to code refactoring (Ouni et al. 2013) represent some examples of software design activities that have been solved from a multi-objective perspective.

Within the field of search-based software design, software architecture optimisation (Aleti et al. 2013) encompasses those works devoted to automatically specifying,

re-designing, discovering or deploying software architectures. Recently, these tasks have also being addressed with multi-objective evolutionary approaches. Grunske (2006) presented a first analysis of the applicability of a multi-objective strategy facing several specific non-functional requirements to the software architecture refactoring problem. In this case, no implementation was proposed.

The generation of a set of alternative architectural specifications by means of a specific multi-objective genetic algorithm is proposed in Räihä et al. (2011). Here, a structural model, specified as a class diagram representing a low level architecture, is derived from its corresponding behavioural specification, modelled in terms of sequence diagrams, which are extracted from the software requirement specification and drawn as use cases. Simultaneously, 4 design metrics related to modifiability and efficiency are optimised in the process. NSGA-II is the algorithm chosen to perform the architectural deployment in Koziolek et al. (2013), looking for a trade-off between availability, performance and cost. Here, the optimisation problem consists of the optimal allocation of components into servers, including the server configuration and component selection.

Although NSGA-II stands out as the preferred algorithm in all the SBSE fields, some comparative studies have recently appeared to empirically decide the most appropriate MOEA for a given design problem. With reference to software architecture deployment, Li et al. (2011) compares SPEA2, NSGA-II and SMS-EMOEA (S Metric Selection Evolutionary Multiobjective Optimisation Algorithm) to obtain the best allocation of software components in terms of CPU utilization, cost and latency. Further, there are some recently proposed studies related to other problem domains. For example, in Zhang et al. (2013) the performance of several variants of NSGA-II were analysed for the priorisation of requirements, considering up to 5 objectives. In Assunção et al. (2014), the authors propose a comparison between SPEA2, NSGA-II and PAES (Pareto Archived Evolution Strategy) to deal with 2 and 4 objectives in the context of software testing. Only del Sagrado et al. (2015), Durillo et al. (2011), and Luna et al. (2014) seem to consider a greater variety of metaheuristics, even though they are actually applied to bi-objective problems, like the selection of requirements or the project scheduling problem.

Classical MOEAs still are the most used algorithms in SBSE for multi-objective problems, though some of these problems really seem to require more than 2 or 3 objectives to be optimised (Sayyad and Ammar 2013). Many-objective approaches have received less attention and their applicability has been only explored in a few proposals, mainly focusing on one specific algorithm for a very certain task. In Sayyad et al. (2013) IBEA was chosen to optimise 5 design characteristics of product lines. Kalboussi et al. (2013) presented a preference-based evolutionary method to deal with 7 objectives of a testing problem, whereas Mkaouer et al. (2014) has recently proposed an interesting solution to the software refactoring, using NSGA-III to simultaneously optimise 15 objectives.

3 Software Architecture Discovery

3.1 Conceptual Framework

The ISO Std. 42010 (ISO 2011b) defines the architecture of a software system as "the fundamental concepts or properties of a system in its environment embodied in its elements, relationships, and in the principles of its design and evolution". A software architect is in charge of conceiving a system that meets the expected functional requirements, as well as the non-functional requirements like performance or maintainability, at a high

level of abstraction. A commonly used approach for developing the software architecture specification is to follow a component-based design, which advocates for the construction of independent, interrelated software artefacts, mostly conceived in favour of reuse. In a component-based architecture, a *component* represents a block of highly cohesive functionality, whose interactions are specified by means of *provided* and *required interfaces* (Szyperski 2002). *Connectors* are links between provided and required interfaces, so they tie a component that supplies some services to another component demanding them.

The tasks related to the architectural design deal with aspects like the understanding, reuse, construction, evolution, analysis and management of the software system (Garlan 2000). From the beginning, an architectural specification is not only required to provide an abstract description of the system, but also to check its consistency and conformance to quality attributes. Nevertheless, the techniques currently used to evaluate software architectures are mostly based on the expert's opinion (Dobrica and Niemela 2002). Actually, the definition of design metrics at the architectural level is still a great challenge, where the ISO Std. 25000 (ISO 2011a) can play an important role as it can serve as a framework to provide a precise guidance on how an architectural model should be assessed.

Ideally, the architecture specification of the system should be maintained throughout the entire development process, evolving as the software does. Frequently, uncontrolled changes in the requirements or the inclusion of a new functionality are directly reflected in the most concrete artefacts, like source code, leading to out-of-date, incoherent, and useless software documentation. As a result, understanding the original architecture becomes a complicated task (Ducasse and Pollet 2009). Although diverse methods have been proposed to recover the architectural specification, most of them start with the source code, meaning that the recovery process can only be conducted once the system implementation is completed. If code is not available, the discovery of the original architecture could only be done manually in accordance to the analysis models. As this process strongly relies on the expertise of software engineers, providing semi-automatic approaches could serve to assist them, especially in exploring different design alternatives in terms of the expected quality attributes.

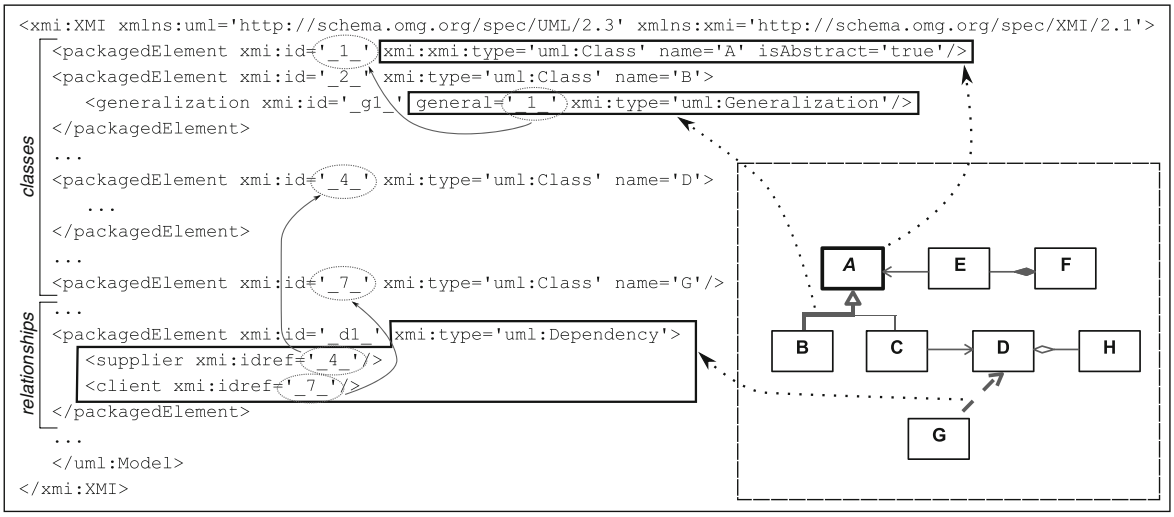


Fig. 1 General outline of the XMI file

3.2 The Search Problem

In Ramírez et al. (2015), the discovery of software architectures is presented as a search problem, where an analysis model represented in form of a class diagram (OMG 2010) saved in XMI format provides the input information in terms of classes and their relationships (see Fig. 1). In this case, the search problem is solved by a single-objective evolutionary algorithm, which is able to generate one single candidate solution representing a high-level architectural specification in terms of components, interfaces and connectors, and depicted by a component diagram. The discovery process is guided by the following rules:

- A component is represented by a cohesive group of classes. The search algorithm should be able to discover the groups of classes that best integrate the different functionalities of the interacting system. Relationships between components are mapped into interacting interfaces. Two constraints should be satisfied in this regard: *any class should belong to only one component*, and *the resulting architectural specification should not contain any empty element*.
- Once the classes are divided up among components, directed relationships between pairs of classes, each belonging to a different component, would represent a candidate interface. The types of the relationships are those defined by UML 2: associations, dependencies, aggregations, compositions and generalisations. Here, two exceptional cases could cause the interface not to be created: either the navigability of the relationship is unspecified, or the relationship represents a data abstraction modelled by a generalisation. Two additional constraints should be also considered: *any component should define at least one interface*, and *mutually dependent components, i.e. a pair of components each one providing services to the other, should not be created*.
- A connector is a link between a pair of required-provided interfaces belonging to different components.

4 Optimisation Approach for the Discovery Process

According to Aleti et al. (2013), the proposed approach can be classified as an architecture optimisation method, where a search-based algorithm is applied on a number of architectural artefacts in order to get the best trade-off between certain quality attributes. More specifically, our evolutionary model falls into the category of multi-objective approaches (*dimensionality* category), making use of a high-level technique, i.e. metaheuristics, to find approximate solutions (*optimisation strategy*). Similarly to other methods, constraint handling techniques are applied, and UML 2 is used as a notation for the architectural specification. What distinguishes our approach from other existing architecture optimisation methods are those aspects related to the problem domain, i.e. quality attributes and decision variables (*degrees of freedom*). The discovery of software architectures was recently proposed as a search problem in Ramírez et al. (2015), being addressed from a single-objective perspective.

The architectural specification is not constructed from scratch during the search-based discovery process. Instead of being derived from requirements, the identification of the system functional blocks and their abstract specification are carried out from an earlier analysis model. A similar approach is the decomposition process proposed by Lutz (2001), where an evolutionary algorithm is used to find the best distribution of classes into modules. This proposal is founded on clustering, so the system structure is viewed as a graph

and, besides, valuable analysis information, such as the presence of abstract artefacts or the existence of different types of relationships among the classes, is omitted during the search. Nonetheless, the proposed procedure is not intended to precisely simulate what the engineer would do to build the architecture, e.g. defining interacting interfaces. Similarly, software modularisation (Praditwong et al. 2011) deals with the organisation of code into packages and modules, working at a lower level of abstraction. Although the three optimisation approaches share the idea of searching the optimal organisation of some kind of software artefacts in other units of construction, each problem defines its own objectives and constraints, leading to several differences among them regarding their respective fitness landscapes.

The elements that constitute the evolutionary algorithm addressing the discovery process are introduced next. Firstly, the encoding and the initialisation are explained. Secondly, the objective functions and the genetic operator, composed by five different mutation procedures, are presented. Crossover is not applied because the evolutionary model presented here adopts the evolutionary programming paradigm.

4.1 Encoding and Initialisation

The individuals of the population represent design solutions in the form of a component-based software architecture. A simple genotype/phenotype mapping constitutes an essential element in favour of interpretability. Additionally, since no prior assumption of the architectural structure to be returned is made, a flexible encoding is also required in order to be able to manage architectural solutions having different number of components. There-

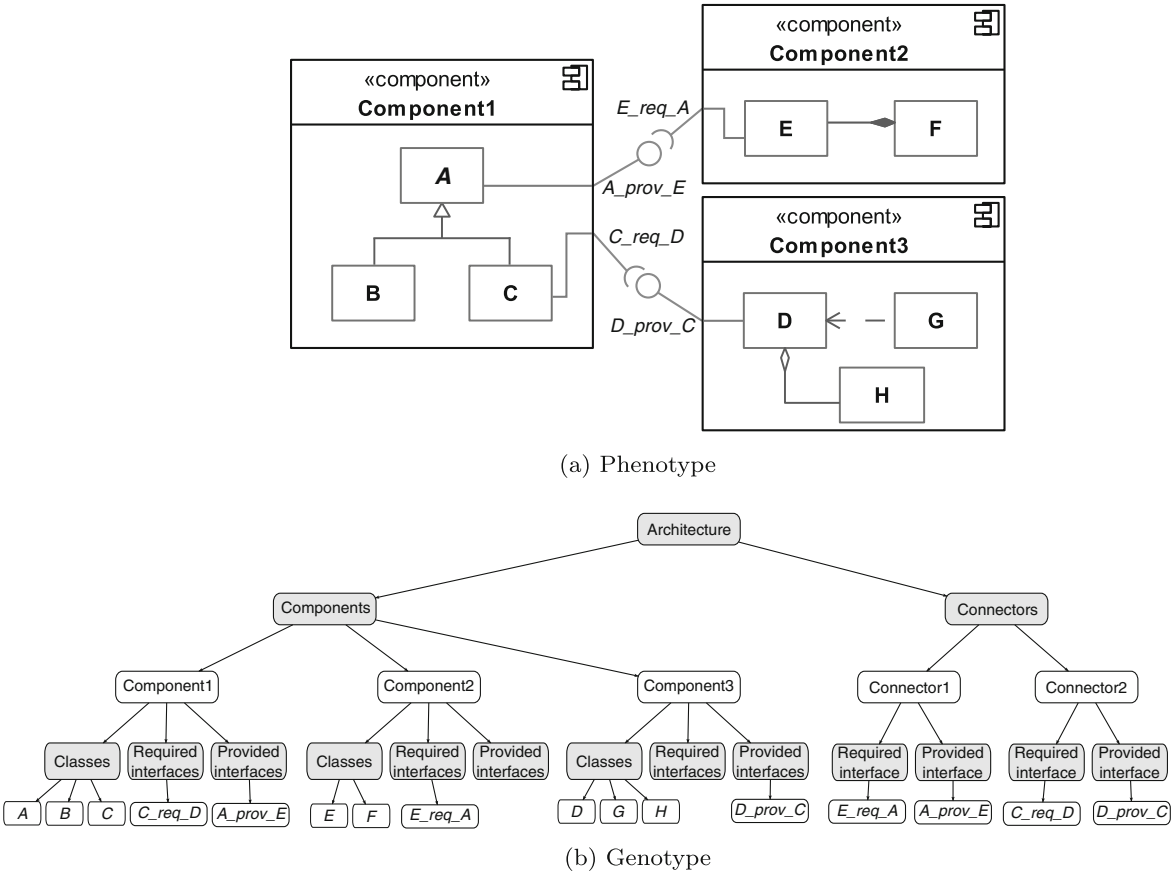


Fig. 2 The component diagram in (**a**) is mapped into the tree structure depicted in (**b**)

fore, a tree structure is used to encode each architectural model, allowing a comprehensible decomposition of the involved software artefacts as established in the problem statement. As depicted in Fig. 2, shaded nodes stand for mandatory elements, whereas the rest of nodes vary from one individual to another, i.e. components and connectors and their internal composition, vary from one individual to another. Additional data about the input class diagram, such as the original relationships between classes and its abstractness are kept by the algorithm. Since this information does not vary from one individual to another, it is not explicitly represented in the genotype.

The initialisation process randomly generates component-based architectures for a given input analysis model. For each solution, a random number of components is chosen between a minimum and a maximum value that is configurable by the software architect. All the classes within the input class diagram are arbitrarily distributed among components. As a constraint none of these components can be empty. Interfaces and connectors are also set considering the existing relationships between classes belonging to different components. The obtained individuals do not need to strictly satisfy all the constraints related to the interaction between components, since they are not checked at this point in order to promote a quicker and more flexible initialisation step. Unfeasible individuals should be turned into feasible ones or be discarded as the evolution elapses, right after the application of the mutation operator and the selection mechanisms.

4.2 Evaluation Objectives

Nine software quality metrics are mapped into their respective objective functions, which serve to guide the evolutionary process. These metrics quantify diverse non-functional requirements related to the maintainability of the software system, which stands out as one of most important quality criteria for component-based software architectures. The software product quality model specified by the ISO Std. 25000 (ISO 2011a) defines *maintainability* as "the degree to which the software product can be modified", where modifications "may include corrections, improvements or adaptation of the software to changes in the environment, and in requirements and functional specifications". This standard document defines the sub-characteristics into which the maintainability can be decomposed to effectively measure a number of related design aspects. The specific maintainability terms applicable to the nature of the addressed problem are the following:

- *Modularity*, which is defined as "the degree to which a system is composed of discrete components such that a change to one component has minimal impact on other components".
- *Reusability*, which establishes "the degree to which an asset can be used in more than one software system or in building other assets".
- *Analysability*, which determines "the degree to which the parts of the software to be modified can be identified".

All these terms need to be translated into quantifiable metrics that will then be selectively combined to be simultaneously optimised by the evolutionary algorithm. On the basis of the ISO Std. 25000, a review of the literature was conducted to select those quality metrics that best suited an architectural specification and could be directly computed over the architectural solutions. These metrics are explained next.

Intra-modular Coupling Density (icd) This metric was proposed by Gupta et al. (2012) and establishes a trade-off between cohesion and coupling, as shown in (1). icd is defined

as the ratio between internal and external relations in the components. ci_i^{in} represents the number of interactions between classes inside the component i, whereas ci_i^{out} is the number of external relations, i.e. the number of candidate interfaces. Here, the ratio between ci_i^{in} and ci_i^{out} is weighted by the fraction of classes taking part in these relationships. Finally, icd is calculated for the overall architecture as the average of all the icd_i values, all these terms varying in [0,1].

$$icd_i = \frac{\#classes_{total} - \#classes_i}{\#classes_{total}} \cdot \frac{ci_i^{in}}{ci_i^{in} + ci_i^{out}} \quad icd = \frac{1}{n} \cdot \sum_{i=1}^{n} icd_i \tag{1}$$

External Relations Penalty (*erp*) This metric is inspired by the *Interface Violations* metric (Krogmann 2010). erp computes the number of existing relationships between classes belonging to each possible pair of different components, i and j, that could not be defined as candidates interfaces (see (2)). More precisely, these relationships are those associations (as), aggregations (ag) and compositions (co) that do not explicitly specify their navigability, as well as generalisations (ge). A weighted sum (w_x) is calculated to emphasise those relations that strongly harm the overall quality of the architectural solution at the software engineer's discretion.

$$erp = \sum_{i=1}^{n} \sum_{j=i+1}^{n} (w_{as} \cdot n_{as_{ij}} + w_{ag} \cdot n_{ag_{ij}} + w_{co} \cdot n_{co_{ij}} + w_{ge} \cdot n_{ge_{ij}}) \tag{2}$$

Instability (*ins*) The instability metric is intended to create highly independent components, leading to more separated functionality blocks. Although the concept of instability was originally defined by Martin (1994) to evaluate the stability of object-oriented designs, it can be also considered at the architectural level. In this way, the overall instability of the architecture is obtained as the average value of the instability of each single component (ins_i) (see (3)). In order to calculate this metric, the terms ac_i and ec_i need to be first computed. ac_i represents the afferent coupling of component i, i.e. the number of components that require its services, whereas ec_i stands for the efferent coupling, i.e. the number of external components that provide their services to component i (Sant'Anna et al. 2007).

$$ins_i = \frac{ec_i}{ec_i + ac_i} \quad ins = \frac{1}{n} \cdot \sum_{i=1}^{n} ins_i \tag{3}$$

Encapsulation (*enc*) This metric is inspired by the *Data Access Metric* proposed by Bansiya and Davis (2002), which serves to evaluate the encapsulation of one class within an object-oriented design. With the aim of adapting this concept to the architectural level, the encapsulation of each component i, enc_i, is computed as the ratio of its hidden classes, i.e. those that do not participate in any interaction with classes belonging to other components in the architecture, and the total number of contained classes. The overall encapsulation is calculated as the mean of the enc_i values (see (4)).

$$enc_i = \frac{\#inner_{classes}}{\#total_{classes}} \quad enc = \frac{1}{n} \cdot \sum_{i=1}^{n} enc_i \tag{4}$$

Critical Size (*cs*) This metric, presented in Narasimhan and Hendradjaya (2007), intends to minimise the number of large components. To this end, cs counts the number of critical

components with respect to their size (cc_{size}), this value being returned by the $size()$ function and calculated as the percentage of classes comprised by the component in relation to the entire model. A threshold value is required to determine if the corresponding component would be critical or not (see (5)).

$$cc^i_{size} = \begin{cases} 1 \text{ if size}(i) > \text{ threshold} \\ 0 \text{ otherwise} \end{cases} \qquad cs = \sum_{i=1}^{n} cc^i_{size} \tag{5}$$

Critical Link (*cl*) Similarly to cs, cl considers the number of critical components, cc_{link}, with respect to the number of their interactions within the architecture (Narasimhan and Hendradjaya 2007). As can be seen in (6), the links refer to the number of provided interfaces and a threshold should be established.

$$cc^i_{link} = \begin{cases} 1 \text{ if } \#provided\ interfaces_i > \text{ threshold} \\ 0 \text{ otherwise} \end{cases} \qquad cl = \sum_{i=1}^{n} cc^i_{link} \tag{6}$$

Groups/Components Ratio (*gcr*) The gcr metric, presented in (7), has been adapted from the *Component Packing Density* (cpd) metric, defined by Narasimhan and Hendradjaya (2007). cpd determines the ratio between the number of constituents, i.e. those elements that compose the components, and the total number of components in the architecture. Here, the constituents are the groups of connected classes ($\#cgroups$) inside each component.

$$gcr = \frac{\#cgroups}{\#components} \tag{7}$$

Abstractness (*abs*) This measure (Krogmann 2010) is defined as the ratio of abstract classes inside a component, abs_i (see (8)). A global value of the abstractness distribution for a given architectural solution, abs, can be obtained by computing the mean value of abstractness for each component.

$$abs_i = \frac{\#abstract\_classes_i}{\#classes_i} \qquad abs = \frac{1}{n} \cdot \sum_{i=1}^{n} abs_i \tag{8}$$

Component Balance (*cb*) Reducing the presence of critical components is an important factor in software design, though it could lead to an excessive number of tiny components hampering the comprehension of the resulting system structure. cb is proposed in Bouwers et al. (2011) to quantify the balance among components, taking into consideration their respective sizes. It is based on two terms, the system breakdown (sb) and the component size uniformity (csu), as shown in (9), where csu is defined as the Gini Coefficient, i.e. a statistical measure of dispersion. The $volume(c)$ function counts the number of classes per component c, C being the set of components in the architecture. csu varies in the range [0,1], the unity being its optimum value, meaning that all the components have the same size. sb represents a linear function in the range [0,1], which benefits the proximity to a number of components, μ, and penalises those architectures having just one or too many components (ω). Since the problem definition determines the minimum and maximum number of components to be comprised by the candidate architectures, ω should be set to this maximum value, and the theoretical minimum threshold, 1, should be replaced

Table 1 Main characteristics of objective functions

Metric	Min/Max	Quality attribute	Range of values	Design goals
icd	max	modularity	[0, 1]	Small components with high cohesion
erp	min	modularity	[0, $*$]	Large components with low coupling
ins	min	modularity	[0, 1]	Components with few interactions
enc	max	modularity	[0, 1]	Components with hidden classes
cs	min	modularity	[0, n]	Small or medium-sized components
cl	min	modularity	[0, n]	Components with few provided interfaces
gcr	min	reusability	[1, $*$]	Connected classes within each component
abs	max	reusability	[0, 1]	Components with abstract classes
cb	max	analisability	[0, 1]	Equal-sized components

by the minimum number of components. Similarly, the μ value will be equal to the average value between these maximum and minimum values.

$$
\begin{aligned}
sb(n) &= \begin{cases} \frac{n-1}{\mu-1} & \text{if } n < \mu \\ 1 - \frac{n-\mu}{\omega-\mu} & \text{if } \mu < n < \omega \\ 0 & \text{if } n \geq \omega \end{cases} \\
csu(C) &= 1 - Gini(\{volume(c) : c \in C\}) \\
cb(S) &= sb(|C|) \cdot csu(C)
\end{aligned} \tag{9}
$$

Table 1 summarises the main characteristics of the 9 objective functions, including whether they are to be minimised (min) or maximised (max), and a brief description of the design goals being promoted. Each metric is related to maintainability characteristics defined above, and the range of possible values is also provided. The symbol $*$ is used to represent that the upper bound of erp and gcr is dependent on the features of the problem instance, whilst n indicates the number of components within the architecture.

Table 2 shows the list of design goals being promoted ($\uparrow$) or demoted ($\downarrow$) by each metric. The symbol $-$ represents that the corresponding metric has no influence on that goal. As can be seen, most of the trade-offs between metrics are related to the size of components and the way in which components would preferably interact with each other. For example, those metrics that look for decreasing the number of interactions between components, in terms of both interfaces and external relationships, i.e. coupling, tend to demote the internal component cohesion due to an excessive encapsulation of functionalities.

4.3 Mutation Operator

The execution of a crossover operator could produce a significant amount of unfeasible solutions, requiring the implementation of repair methods to satisfy those constraints that had been already checked during the initialisation process. It would be a time-consuming procedure, without which unfeasible individuals could be generated even late in the search, making the convergence to the feasible region of the search space considerably more difficult. Only a domain-specific mutation operator serves to explore the different architectural solutions throughout the evolutionary search (Ramírez et al. 2015). A weighted roulette is used to select one of the five proposed mutation procedures. More specifically, these procedures simulate those architectural transformations that could be applied to a given individual, i.e. the parent, to generate an offspring.

- *Add a component*. A new component is added to the architecture. Creating such a new component requires extracting some classes from other components. More specifically, if a component would contain several unconnected groups of related classes, some of them could be randomly selected to create the new component. Hence, the ultimate aim of this procedure is to improve the distribution of functionalities among components.
- *Remove a component*. This procedure implies that one component in the current architecture will be discarded. In this case, its inner artefacts are arbitrarily distributed among the remaining components. A component having a great number of dependencies with others is a preferable candidate to be removed.
- *Merge two components*. This procedure involves the construction of a new component by taking two existing components from the current architecture. Since the aim of this procedure is to attempt to concentrate functionalities, the first selected component would be one with a large number of external dependencies. Then, a second component is selected at random.
- *Split a component*. This procedure is designed with the aim of avoiding excessively large components. One component is randomly selected and split into two components, forcing an interaction between both of them to remain in the form of a new interface. More precisely, an internal relationship is chosen at random to act as the candidate interface. Next, each inner class is assigned to one component or another depending on how it relates to the interface ending, i.e. classes providing and requiring the service are separated.
- *Move a class*. This procedure executes the simple movement of one class from one component to another. Notice that this procedure does not apply any change in the structure of the solution, so the transformation is done completely at random.

The roulette is dynamically built for each parent, only considering those applicable procedures in each particular case. Thus, if the initial individual is a feasible solution, the mutation procedure will always return a solution without duplicate classes or empty components. The minimum and maximum number of components within the architecture is also preserved. For example, the *split a component* operation cannot be executed on an individual representing an architecture that already contains the maximum number of components. The weights of the roulette can be configured by the software architect. Finally, notice that the existing constraints related to interactions between components have to be considered here. For example, if the obtained individual represents an unfeasible solution, the parent is mutated again until a feasible solution is reached or a maximum number of

Table 2 Design goals and trade-offs between metrics

Design goal	*icd*	*erp*	*ins*	*enc*	*cs*	*cl*	*gcr*	*abs*	*cb*
Small components	↑	↓	↑	↓	–	–	↓	–	↓
Large components	↓	↑	↓	↑	↓	–	↑	–	↓
High cohesion between classes	↑	↓	–	↓	–	–	↓	–	–
Low coupling between components	–	↑	–	–	–	–	–	–	–
Few interfaces per component	↓	–	↑	↑	–	↑	–	–	–
High encapsulation	–	–	↑	↑	–	–	↑	–	–
Distribution of abstract classes	–	–	–	–	–	–	–	↑	–

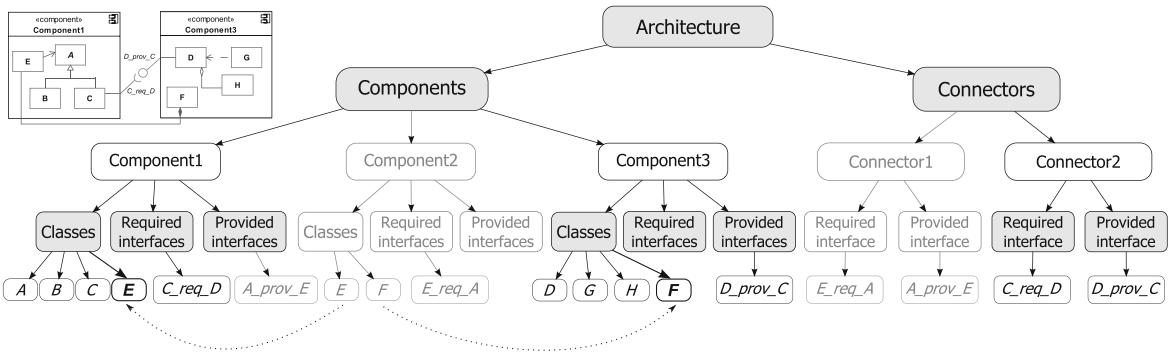


Fig. 3 Example of the mutation operator

attempts is exceeded. In the latter situation, the unfeasible mutant individual could be considered as a valid offspring depending on a probabilistic criterion. In this sense, a dynamic threshold is set at the first generation, and will be decreased along the evolution. If a randomly generated number exceeds the current threshold, the unfeasible individual is added to the offspring pool. On the contrary, the original parent takes his position in the set of descendants.

As a way to illustrate how the mutation operator works, Fig. 3 shows the result of applying the remove component procedure over the solution presented Fig. 2. As can be observed, the second component ($Component2$) within the original individual has been removed, and its classes randomly reallocated. In this example, class E is moved to $Component1$, whilst class F is reallocated to $Component3$. As a result, since classes A and E are now part of the same component, the existing interaction between them becomes internal. No more interfaces are created to link classes E and F, since the navigability of their composition is unspecified.

5 Experimental Framework

All the algorithms and the experimental framework have been coded in Java. Additionally, some public Java libraries have been used to facilitate the implementation. SDMetrics Open Core[1] provides support for parsing XMI files, allowing the extraction of information from the analysis models created with the MagicDraw tool.[2] Preprocessing tasks and internal data structures have been implemented using the Datapro4j library.[3] The evolutionary algorithms have been written using JCLEC (Ventura et al. 2008). Experiments were run on a HPC cluster of 8 compute nodes with Rocks cluster 6.1 x64 Linux distribution, where each node comprises two Intel Xeon E5645 CPUs with 6 cores at 2.4 GHz and 24 GB DDR memory.

In this section, the comparison methodology is presented, including the performance measures used to evaluate the evolutionary algorithms. The parametrisation and set-up, as well as the problem instances used, are also introduced next.

[1] http://www.sdmetrics.com/OpenCore.html

[2] http://www.nomagic.com/

[3] http://www.uco.es/grupos/kdis/datapro4j

5.1 Methodology

To assess the validity of the experiments, every algorithm has been executed 30 times with different random seeds. The 9 proposed design metrics have been combined in groups of 2, 4, 6, 8 and 9 objectives, which make a total of 256 combinations. After executing all these combinations, it is possible to analyse how the selection of a particular subset of metrics, including its cardinality, can affect both the evolutionary performance and the type of obtained architectural solutions.

Regarding the comparison study from an evolutionary perspective, two quality indicators, hypervolume (HV) and spacing (S) have been considered (Coello Coello et al. 2007). HV calculates the hyper-area covered by the Pareto set, whereas S measures the spread of the front. It should be noted that HV requires an objective space in the range [0,1], so the normalisation approach proposed in Luna et al. (2014) is applied after executing the algorithms. More specifically, a reference Pareto front (RPF) is built up considering all the solutions found after executing the 8 algorithms for each problem instance. Then, the objective values are normalised taking as reference the worst value and the best value achieved by any solution within the RPF.

Statistical tests (Arcuri and Briand 2011; Derrac et al. 2011) have served to compare the performance of the algorithms under study. The Friedman test, a non-parametric test that allows comparing multiple algorithms over different problem instances, is executed first. At a specific level of significance, $1 - \alpha$, this test establishes the null hypothesis, H_0, denoting that all the algorithms perform equally well. If H_0 is rejected, i.e. significant differences are detected, a post-hoc procedure has to be carried out in order to properly determine those statistical differences between the considered algorithms. Here, the Holm test is performed when H_0 cannot be accepted, a control algorithm being compared to the rest of algorithms. Additionally, the Cliff's Delta test has been executed to assess the magnitude of the improvements using an effect size measurement (Arcuri and Briand 2011). This test performs pairwise comparisons to determine whether such an improvement should be considered negligible, small, medium or large on the basis of specific thresholds (Romano et al. 2006).

Some other measures have been compiled as well in order to precisely state the discussion of results beyond the evolutionary performance. Hence, aspects like the execution time, the size of the Pareto set, or the set of non-dominated solutions are also scrutinised to properly analyse the behaviour of each algorithm concerning the software architect's expectations.

5.2 Parameter Set-up

Before entering into the details of the value assignment to specific parameters of each algorithm, there are some related aspects to be addressed respecting constraint handling and mating selection. On the one hand, notice that constraint handling techniques need to be applied in different ways. A first scenario refers to the way invalid solutions are evaluated. In the case of the algorithms SPEA2, MOEA/D, GrEA, IBEA and HypE, the worst possible fitness value is assigned to every unfeasible individual. A constrained version of NSGA-II was already presented in the original work by Deb et al. (2002), in which feasible solutions were promoted according to a specific comparison method. The same approach has been adopted for ϵ-MOEA, since this algorithm does not declare any fitness function. These two mechanisms guarantee that unfeasible individuals will not prevail over feasible individuals in any selection process.

Table 3 Parameter set-up

Common parameters	
Population size (2, 4, 6, 8, 9 objectives)	100, 120, 126, 330, 495
Max. evaluations (2, 4, 6, 8, 9 objectives)	10,000, 15,000, 20,000, 66,000, 99,000
Min-max. components	2–8
Mutator weights	$w_{add} = 0.2$, $w_{remove} = 0.3$, $w_{merge} = 0.2$, $w_{split} = 0.1$, $w_{move} = 0.2$
erp weights	$w_{as} = 2$, $w_{ag} = 3$, $w_{co} = 3$, $w_{ge} = 5$
cs threshold	0.3
cl threshold	8
SPEA2 parameters	
Parents selector	Binary tournament
External population size	50
k-th neighbour	12
MOEA/D parameters	
Neighbourhood size (τ)	8
Max. replacements (Nr)	2
H (2, 4, 6, 8, 9 objectives)	99, 7, 4, 4, 4
ϵ-MOEA parameters	
ϵ values	$\epsilon_{icd} = 0.05$, $\epsilon_{erp} = 5.00$, $\epsilon_{ins} = 0.05$, $\epsilon_{enc} = 0.05$ $\epsilon_{cs} = 1.00$, $\epsilon_{cl} = 1.00$, $\epsilon_{gcr} = 0.10$, $\epsilon_{abs} = 0.05$, $\epsilon_{cb} = 0.05$
GrEA parameters	
Number of divisions (div)	12
IBEA parameters	
Scaling factor (k)	0.05
HypE parameters	
Number of sampling points (M)	10,000

On the other hand, regarding the mating selection phase in the case of MOEA/D, it usually takes two individuals from the neighbourhood of each member of the population to act as parents and generates a unique descendant. However, given that only one mutator operator is executed, just one neighbour is selected to generate the offspring.

The specific configuration parameters of the selected algorithms have been set up according to their respective authors, whilst the domain-specific configuration is identical in all cases. If required, some preliminary experiments have been performed to properly adapt their parameters to the specific needs of the problem domain, e.g. the hypercube lengths in ϵ-MOEA. The final parameter set-up is shown in Table 3. Notice that different values are set for the population size and the maximum number of evaluations to deal with the growing complexity of the optimisation problem when increasing the number of objectives from 2 to 4, 6, 8 and 9. Regarding the population size, it should be noted that MOEA/D requires a uniformly distributed set of weighted vectors, whose size is controlled by the number of objectives and the parameter H. Therefore, the number of weighted vectors obtained for

Table 4 Problem instances and their characteristics

Problem	*#Classes*	*#Relationships*					*#Interfaces*
		As	De	Ag	Co	Ge	
Aqualush	58	69	6	0	0	20	74
Borg	167	44	109	36	38	90	300
Datapro4j	59	3	4	3	2	49	12
ICal4j	190	36	4	3	11	161	70
Java2HTML	53	20	66	15	0	15	170
JSapar	46	7	33	21	9	19	80
JXLS	96	60	10	10	9	45	136
Logisim	253	113	19	46	25	137	248
Marvin	32	5	11	22	5	8	28
NekoHTML	47	6	17	15	18	17	46

each number of combined objectives has determined the population size for the rest of algorithms. For the remaining parameters, they are related to the problem domain, including the minimum and maximum number of components to be considered for the architecture, the parameters required to calculate the objective functions (see Section 4.2) and the weight values related to the mutation procedures (see Section 4.3).

5.3 Problem Instances

The experimental study has been carried out over 10 diverse problem instances, each one representing a system design with different complexity in terms of both the number of classes and the interoperability among them. Table 4 provides the complete list of their respective characteristics. The number and types of relationships are enumerated in the column *#Relationships*, where the different types are broken down in turn as prescribed by UML 2: associations (*As*), dependencies (*De*), aggregations (*Ag*), compositions (*Co*) and generalisations (*Ge*). The last column (*#Interfaces*) indicates the number of candidate interfaces, i.e. the number those relationships whose navigability has been explicitly specified.

Most of these instances have been taken from actual real-world software systems working on diverse application domains. Only *Aqualush*[4] is a benchmark used for educational purposes. Some of the software specifications can be accessed from the Java Open Source Code Project[5], whereas *Datapro4j*, *iCal4j*[6] and *Logisim*[7] are available at their respective websites.

[4] http://www.ifi.uzh.ch/rerg/research/aqualush.html

[5] http://www.java-source.net/

[6] http://www.sourceforge.net/projects/ical4j/

[7] http://www.sourceforge.net/projects/logisim/

6 Analysis of Results

In this section, we analyse the results from the perspective of the evolutionary performance. More specifically, we focus on the scalability of the selected algorithms regarding the two quality indicators mentioned in Section 5.1, hypervolume and spacing.

6.1 Analysis of 2- to 4-objective Problems

Table 5 shows the overall ranking obtained by the Friedman test for each selected algorithm in terms of hypervolume. A value represents the arithmetic mean of the ranking positions determined considering all the possible combinations for the given number of objectives. Notice that the lowest values are the best. Their corresponding standard deviation is shown accordingly. Similarly, Table 6 shows the summary of the results for the spacing indicator. Complementary information about the results is reported in the Appendix.

Regarding the HV for 2-objective problems, NSGA-II, ϵ-MOEA and HypE achieve the best ranking positions, obtaining significant differences with the rest of algorithms. On the contrary, SPEA2 and MOEA/D show a better behaviour in terms of the spacing indicator. IBEA and NSGA-III achieve good spacing values only for certain combinations of objectives, which is reflected in a higher standard deviation. Notice that simultaneously having good scores in terms of both hypervolume and spacing becomes a great challenge. Even though a multi-objective evolutionary algorithm tries to find a compromise between the convergence towards a set of non-dominated solutions and the spread of the resulting front, its specific characteristics could encourage the promotion of one criterion at the expense of the other. Specific many-objective approaches do not exhibit any superiority yet, obtaining the best ranking positions for only a few problems.

As for the combinations of metrics, it can be noted that their respective optimisation problems may imply several differences in the performance of the algorithms. Since the software architect could be interested in different aspects of the same quality criteria, e.g. several design metrics are related to modularity, the selection of the specific metrics is also an important decision with respect to the expected architectural solutions. For those problems where erp, gcr, cs or cl are jointly optimised, all the algorithms have achieved similar values in terms of both quality indicators. For example, the selection of erp and cl ends in a tie between SPEA2, NSGA-II, MOEA/D and ϵ-MOEA. Moreover, there are no statistical differences when comparing these algorithms to GrEA and HypE, because all the algorithms obtain either a unique solution or an equivalent Pareto set. In the former case,

Table 5 Mean and standard deviation of Friedman rankings for hypervolume

Algorithm	2 objectives	4 objectives	6 objectives	8 objectives	9 objectives
SPEA2	4.35 ± 0.78	4.66 ± 0.86	5.64 ± 0.29	6.33 ± 0.28	6.70 ± 0.00
NSGA-II	2.02 ± 0.59	1.39 ± 0.40	1.92 ± 0.54	2.66 ± 0.46	2.90 ± 0.00
MOEA/D	3.90 ± 0.81	4.41 ± 0.68	3.75 ± 0.47	3.48 ± 0.43	3.30 ± 0.00
ϵ-MOEA	3.88 ± 1.12	2.78 ± 1.15	1.51 ± 0.55	1.08 ± 0.09	1.00 ± 0.00
GrEA	4.86 ± 0.79	5.30 ± 0.75	5.80 ± 0.41	5.83 ± 0.11	5.50 ± 0.00
IBEA	7.21 ± 0.31	7.64 ± 0.15	7.76 ± 0.06	7.79 ± 0.05	7.90 ± 0.00
HypE	3.04 ± 0.58	3.20 ± 0.75	3.20 ± 0.55	2.82 ± 0.31	2.80 ± 0.00
NSGA-III	6.75 ± 0.49	6.62 ± 0.21	6.49 ± 0.16	6.01 ± 0.29	5.90 ± 0.00

Table 6 Mean and standard deviation of Friedman rankings for spacing

Algorithm	2 objectives	4 objectives	6 objectives	8 objectives	9 objectives
SPEA2	3.65 ± 1.25	2.09 ± 1.08	1.10 ± 0.17	1.37 ± 0.37	1.70 ± 0.00
NSGA-II	4.00 ± 0.82	4.59 ± 0.91	2.77 ± 0.85	1.97 ± 0.34	1.80 ± 0.00
MOEA/D	3.23 ± 1.10	3.16 ± 0.79	5.21 ± 0.50	5.69 ± 0.23	5.30 ± 0.00
ϵ-MOEA	5.58 ± 1.04	3.02 ± 1.49	3.41 ± 0.69	4.06 ± 0.37	4.60 ± 0.00
GrEA	5.37 ± 0.52	6.78 ± 0.45	7.29 ± 0.17	7.28 ± 0.14	6.60 ± 0.00
IBEA	5.73 ± 1.51	7.43 ± 0.42	7.60 ± 0.04	7.63 ± 0.07	7.80 ± 0.00
HypE	4.09 ± 0.72	4.91 ± 0.95	5.11 ± 0.82	4.88 ± 0.57	5.00 ± 0.00
NSGA-III	4.35 ± 1.45	4.02 ± 1.09	3.53 ± 0.51	3.13 ± 0.16	3.20 ± 0.00

every approach finds an optimal solution, since the objectives are not strongly opposed. In the latter case, all the algorithms have some trouble in avoiding local optima. In constrast, the combination of erp and cl metrics with icd, abs, cb or enc implies the formulation of a more complex optimisation problem, where the dominance between solutions is harder to achieve. Therefore, the resulting Pareto set will be comprised of more diverse solutions, where both objectives are satisfied in different ways.

As the number of objectives increases up to four, differences between them become more evident. It is worth noticing that SPEA2 and NSGA-II are still the best algorithms regarding spacing and hypervolume, respectively, though their rankings for the other indicator have increased. As can be seen in Tables 5 and 6, ϵ-MOEA and NSGA-III have improved their overall rankings for both indicators, whereas the rest of algorithms do not experience great changes or their performance even decreases, as in the case of IBEA and GrEA.

MOEA/D and ϵ-MOEA obtain the best ranking for some specific combinations of objectives. For example, MOEA/D has shown good spacing values when cs, cl or ins are considered. Similarly, ϵ-MOEA seems to optimise certain objectives, e.g. icd, ins, abs or cb, more accurately than NSGA-II in terms of HV. These 4 objectives promote the functional distribution among components. On the contrary, if metrics like erp, gcr, enc and cl are selected to define the search problem, the number of components within the architectural solution tends to be minimised as a way to reduce their interactions. In general, ϵ-MOEA provides a better performance than NSGA-II when the optimisation problem is more complex from a design perspective, as the latter tends to promote the discovery of an excessively monolithic architecture.

Table 7 compiles the outcomes obtained by the different algorithms in terms of their performance when a low number of objecives is considered to address the problem under study.

6.2 Analysis of 6- to 9-objective Problems

When algorithms optimising 6 or more objectives are compared, the Friedman test returns significant differences for a large number of combinations, many of them including at least one objective that promotes a design goal that tends to be demoted by the rest of objectives. Some initial tendencies observed for 4-objective optimisation problems are now clearly brought out. To illustrate this point, Fig. 4 depicts the percentage of combinations for which each algorithm reaches the best ranking for hypervolume and spacing, respectively. As can

Table 7 Summary of the evolutionary performance for 2- and 4-objective problems

Algorithm	Main findings
SPEA2	• Ability to maintain a wide-spread Pareto front due to the use of density information • Some difficulties to obtain competitive values for the hypervolume
NSGA-II	• Good scalability with respect to hypervolume • Variable behaviour in terms of spacing, especially when increasing to 4 objectives • The nondominated sorting approach has a strong influence on the evolution
MOEA/D	• Spacing values reflect effective exploration of multiple search directions using weight vectors • Low ranking positions for both 2- and 4-objective problems in terms of HV
ϵ-MOEA	• Variable behaviour in terms of HV when optimising combinations of 2 metrics • Better HV than other many-objective approaches like HypE or NSGA-III for 4 objectives • ϵ-dominance contributes to enhance the diversity of the Pareto front
GrEA	• Poor performance regardless of the number and combination of metrics • Similar to NSGA-II regarding S, since the sorting approach strongly promotes convergence
IBEA	• Ability to reach reasonably good spacing values for specific 2-objective combinations • Poor performance when dealing with 4 objectives. • ϵ-indicator is a non-discriminatory criterion to guide the search towards promising solutions
HypE	• Lower HV values than expected regardless of the number of objectives • Good trade-off between HV and S for 2 objectives • Excessive promotion of solutions contributing to HV at certain steps of the evolution • Some regions of interest can remain unexplored by the optimisation process
NSGA-III	• Better performance than NSGA-II only in terms of S due to the use of reference points • Overemphasis on promoting diversity leads to a low performance in HV ranking values

be noted, as the number of objectives increases, ϵ-MOEA emerges as the best algorithm for HV, outperforming NSGA-II, and SPEA2 is the best alternative in terms of spacing, especially when dealing with more than 6 objectives.

As can be seen from Tables 5 and 6, SPEA2, NSGA-II, GrEA and IBEA report worse HV ranking values than those obtained for 2 and 4 objectives. On the contrary, MOEA/D, ϵ-MOEA, HypE and NSGA-III have experienced an improvement regarding this indicator, though ϵ-MOEA and HypE obtain competitive values when dealing with 8 and 9 objectives.

Additionally, the Cliff's Delta test assesses that the pairwise comparisons between the algorithms are statistically significant in most objective combinations [8]. For the 9-objective problem, a total of 56 test executions per quality indicator were required. In this case, differences in the magnitude of the corresponding indicator have been catalogued as large in 49 and 48 out of the total number of executions made for HV and S, respectively. On the one hand, the effect size obtained when comparing ϵ-MOEA in terms of HV against any other algorithm is always greater than 0.7. On the other hand, regarding SPEA2, the test

[8] Tables 10 (HV) and 11 (S) in Appendix show the results obtained by the Cliffs Delta test for the 9-objective combination. The full results in raw format for all the combinations are available at http://www.uco.es/grupos/kdis/sbse/RRV15

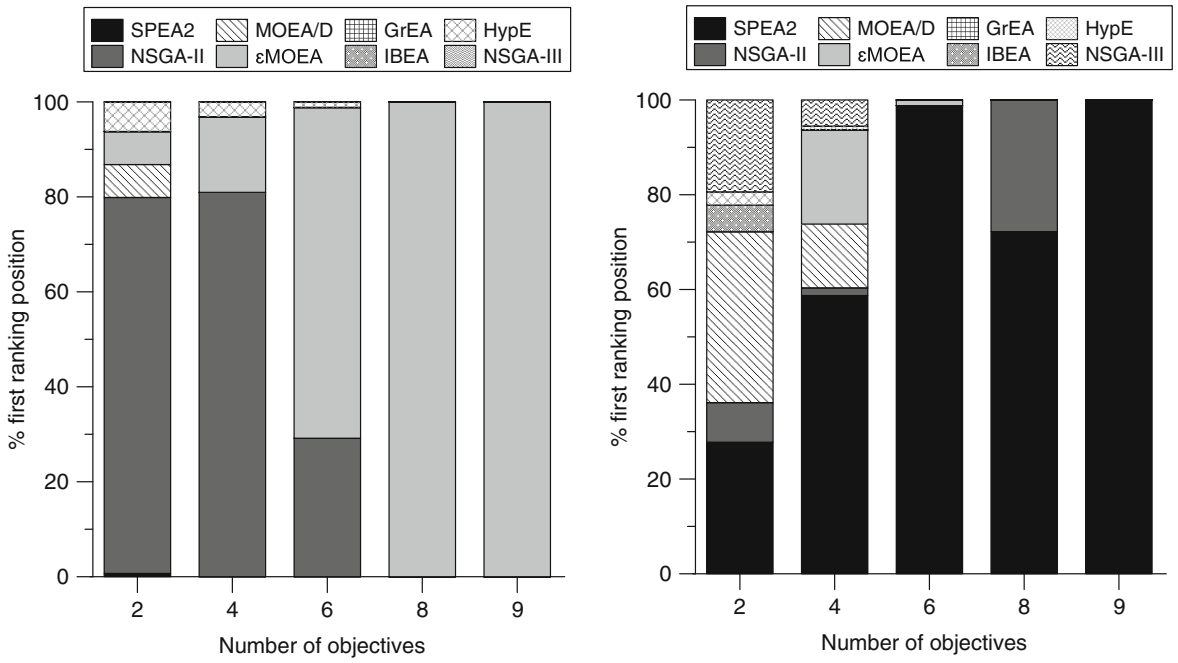


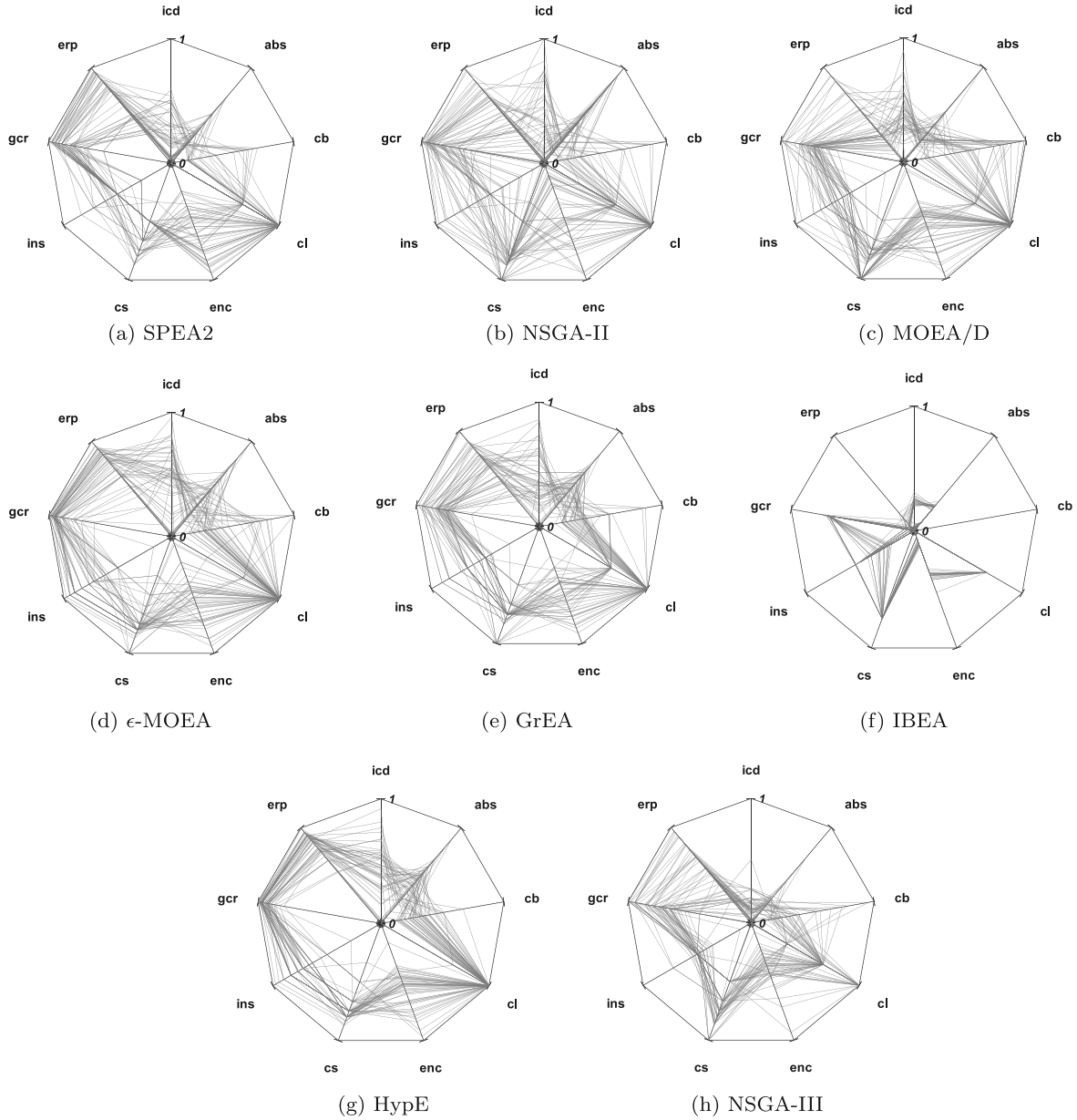
Fig. 4 Percentage of first ranking positions in the Friedman test for hypervolume and spacing

reports an effect size greater than 0.6 for all its comparisons regarding S, except when facing NSGA-II.

Concerning the performance of all the algorithms with more than 4 objectives, the number of objectives has a deeper impact than the set of selected metrics. As can be observed from the experimental results, the ranking values obtained by each algorithm are very similar regardless of the metric subset being optimised. A progressively decrease of the standard deviation is observed in Tables 4 and 5, whereas the number of different algorithms achieving the top ranking position is lower for 4 or 6 objectives than for 2 (see Fig. 4), even when a larger number of objective combinations was generated. Notice that the specific subset of metrics, as well as the need for a different number of objectives, relies on the software architect's judgement. A detailed analysis on the influence of these aspects is provided in Section 7.2.

Some additional remarks can be pointed out with respect to the joint optimisation of 9 objectives. For instance, Fig. 5 shows an example of the Pareto front obtained by each algorithm for a representative problem instance, *Aqualush*. Notice that the objective space is normalised and all the objectives must be maximised. For the sake of readability, a maximum number of 50 solutions is depicted. The solutions have been randomly selected from the set of all the solutions generated in different executions. Those solutions reaching the global minimum and maximum value of each objective are also included in order to identify the worst and best values obtained.

If we look at how lines are distributed, it can be seen that some algorithms tend to focus their search on specific values of particular objectives, while others can reach a wider range of trade-offs among them. More specifically, SPEA2 is unable to reach a proper trade-off among all the metrics, icd, abs and cb usually being optimised worse than erp, gcr, enc and cl. On the contrary, NSGA-II shows great ability to simultaneously deal with all the objectives. In this case, a wide spectrum of solutions is returned, some of them being are able to reach the best values for several objectives. Regarding MOEA/D, most of the obtained



(a) SPEA2 (b) NSGA-II (c) MOEA/D

(d) ϵ-MOEA (e) GrEA (f) IBEA

(g) HypE (h) NSGA-III

Fig. 5 An example of the Pareto front obtained for the *Aqualush* problem instance

solutions lie on intermediate ranges, gcr, cs and cl being frequently promoted to the detriment of icd and abs. This algorithm is able to obtain great diversity of solutions in terms of erp, enc and cb. Focusing on ϵ-MOEA, it can reach competitive values for all the objectives. Its ability to provide several alternatives for icd, enc and cb is noteworthy, whereas good values are frequently reached for gcr and cl. GrEA and HypE behave similarly. Both algorithms experience difficulties to optimise abs and cb, getting only low or medium values. Besides, some solutions tend to concentrate on certain values of the design metrics, generating very similar solutions. HypE converges disproportionally towards erp, gcr and cl. Solutions returned by IBEA show very poor values for all the objectives. Even cb and cs are usually neutralised in benefit of other metrics, causing the returned solutions to be comprised by components completely unbalanced in size. Finally, NSGA-III has had problems to reach high values of icd and abs, although it is able to provide a wide range of values for erp, gcr, cs and enc.

Table 8 Summary of the evolutionary performance for 6-, 8- and 9-objective problems

Algorithm	Main findings
SPEA2	• Valuable diversity preservation when a larger number of objectives is optimised • Unable to converge to the PF as well as other approaches can
NSGA-II	• Behaviour changes when dealing with 8 and 9 objectives • Overtaken by ϵ-MOEA in terms of HV, but shows an improvement in S • Crowding distance may have a deeper impact on individual survival than the sorting method
MOEA/D	• Exploration of an excessive number of directions without favouring any of them • Not all the subproblems are useful to solve the global problem in a discontinuous space
ϵ-MOEA	• Best algorithm with respect to HV, but the observed S variability still remains • The many-objective approach with the best trade-off between both quality indicators
GrEA	• Low ranking values for both quality indicators • No evidence of improvement as the number of objectives increases
IBEA	• Low ranking values for both quality indicators, like GrEA • Outperformed by all other algorithms
HypE	• Intermediate ranking values for both quality indicators • Equivalent to the corresponding control algorithms for specific combinations
NSGA-III	• Only outperformed by SPEA2 and NSGA-II in terms of S • Ranking positions are still low with respect to HV

Table 8 shows the most relevant aspects related to the behaviour of the different algorithms when dealing with highly dimensional objective spaces.

7 Discussion of Results

Due to the particular nature of the problem, further discussion of the outcomes is still required, not only from the evolutionary perspective, but also in terms of its applicability in tool support and the software architect expectations. Therefore, there are other additional aspects from the experimentation to be observed that can provide relevant information about both the architectural specification and the decision support process. With this aim, the number of non-dominated solutions, the execution time or the dependencies among metrics have been studied. These factors directly affect the number of solutions given to the software architect or the sort of returned architectural solutions.

7.1 On Applicability in Tool Support

Having a wide variety of alternative solutions to choose from is not an ideal scenario for the software architect, even more so if only a few of them could serve to meet his expectations in terms of diversity. In such a case, the number of solutions could be useless to the expert, who would be unable to handle them all during the decision process. This matter is likely to require some further external post-processing to select the most representative solutions. Figure 6 shows the distribution of returned solutions per number of objectives. More precisely, the Y axis represents the mean number of solutions

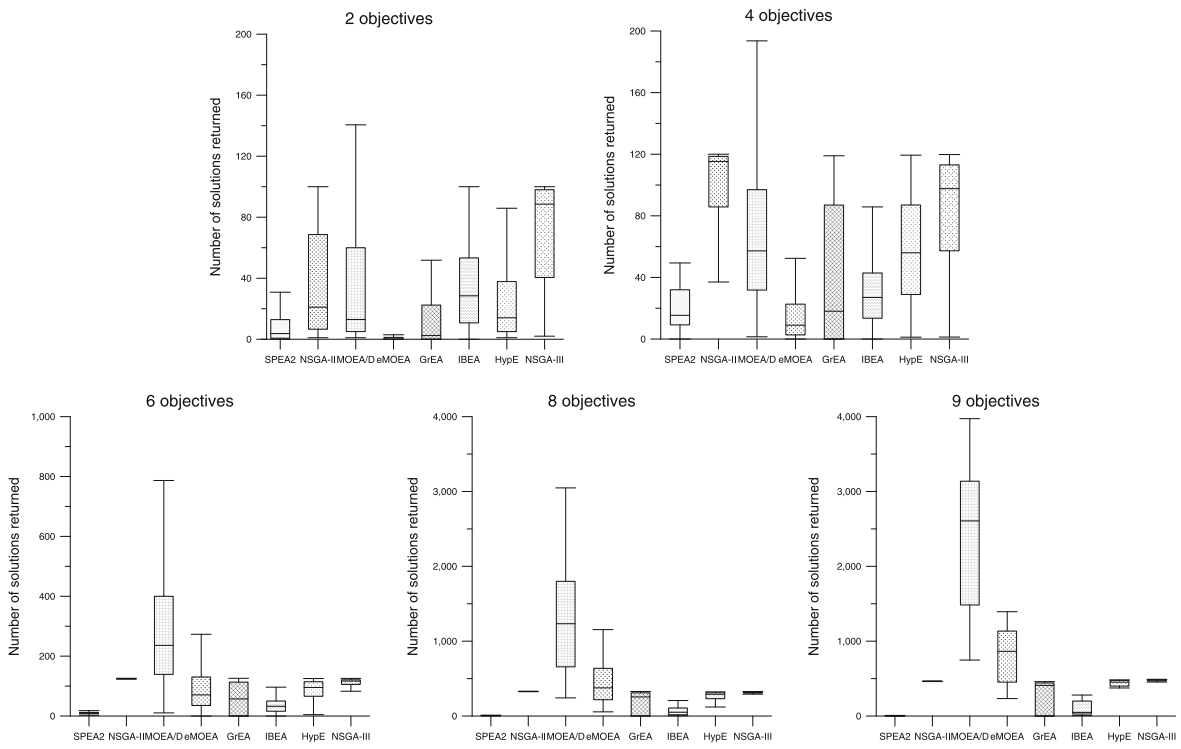


Fig. 6 Distribution of the number of non-dominated solutions composing the Pareto set

found by each algorithm, i.e. the number of non-dominated solutions composing the Pareto set, considering all the possible combinations of design metrics over all the problem instances. As can be seen, the number of objectives has a clear impact on the number of solutions returned. It can be noted that SPEA2 establishes a fixed size for the external population, whereas MOEA/D and ϵ-MOEA do not constrain the size of their archive of solutions. Since NSGA-II, GrEA, IBEA, HypE and NSGA-III do not have any external population, the maximum number of solutions is given by the corresponding population size (see Table 3).

If 2 or 4 objectives are considered, SPEA2 and NSGA-II usually obtain a number of solutions close to the maximum allowed. These algorithms also present some variability in the size of the Pareto set. NSGA-II even shows some difficulties to control such a variability. Having to optimise 6 or more objectives, SPEA2 is unable to fill its external archive with only non-dominated solutions, facing difficulties to find interesting architectures. Finding a more representative set of solutions than the one provided by SPEA2 could benefit the decision making process. NSGA-II always reaches the maximum limit allowed with difficulties to reject equivalent solutions along the search process. MOEA/D provides an excessive number of final solutions for any design problem. Besides, preserving the variety of these solutions is not guaranteed since the maximum number of solutions associated to each search direction is not restricted. The same behaviour could be expected for ϵ-MOEA, but the use of the ϵ-dominance criteria to manage the addition of solutions into the archive clearly serves to implicitly limit its size.

Depending on the combination of objectives, GrEA, IBEA, HypE and NSGA-III can generate as many solutions as the prefixed maximum. As the number of objectives increases, IBEA and GrEA return a lower number of solutions. In contrast, HypE and NSGA-III tend to get closer to their respective maximum limit. In this sense, many-objective approaches provide a better control than NSGA-II and MOEA/D. Nevertheless, only SPEA2 permits setting up the estimated number of solutions that should be returned after the search process.

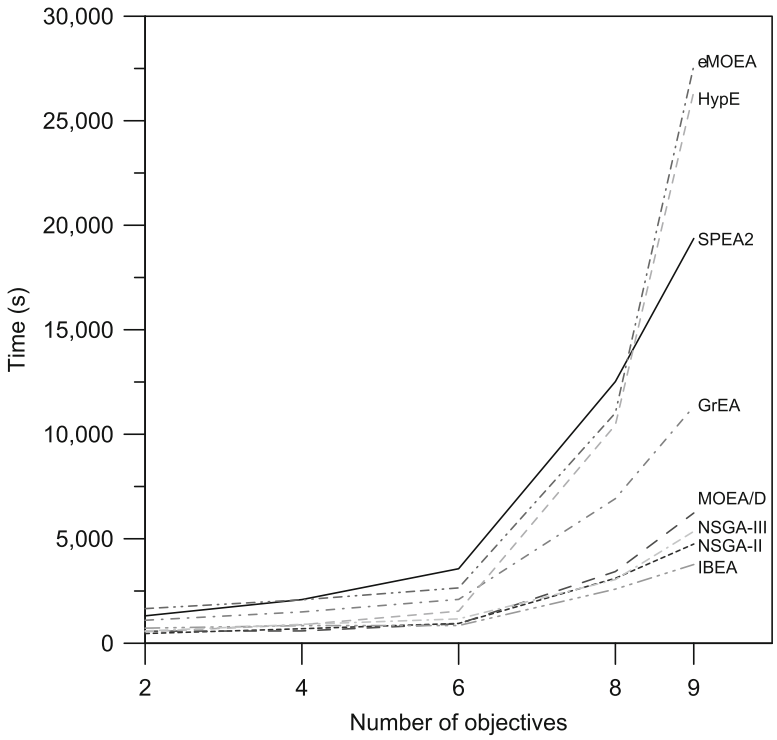


Fig. 7 Average execution time of the evolutionary algorithms

Hence, it is a useful mechanism for the expert, who might be interested in strictly determining the number of solutions that he really wants to check. Notice that GrEA, SPEA2, IBEA and ϵ-MOEA might finish the search without generating any architectural solutions that fulfil the constraints (see Fig. 6). These algorithms have some difficulties in dealing with invalid solutions when optimising the most complex problem instances and certain combinations of objectives. Setting a larger number of iterations would be a first attempt to address this situation. The ability to handle constraints is a relevant factor with respect to its generic applicability to the decision support process.

Another important factor that may affect the usability of search-based approaches is the execution time. Figure 7 shows the scalability of the average execution time of each algorithm in relation to the number of objectives. It can be observed that dealing with many design metrics influences SPEA2, GrEA, ϵ-MOEA and HypE. The density estimator based on clustering proposed by SPEA2, the computation of distance metrics in GrEA, and the hypervolume estimation required by HypE are operations that clearly affect the performance of these algorithms. Besides, SPEA2, GrEA and ϵ-MOEA present some difficulties when removing invalid solutions because they have to execute a large number of attempts during the mutation phase, especially when dealing with the most complex problem instances. Since this kind of instances tends to comprise more interconnected classes, identifying independent components becomes a harder work. In this sense, software engineers should consider if getting high quality solutions could make up for the amount of time spent on the process. On the one hand, ϵ-MOEA seems to be a better choice than HypE or NSGA-II if the architect is mostly interested in high quality solutions. On the other hand, NSGA-II might provide good enough solutions for most of the objective combinations, its execution time being steady and suitable. It is worth mentioning that the computational cost of NSGA-III has proven to be quite similar to NSGA-II, concluding that its execution time has not been significantly influenced by the adjustments proposed to deal with many-objective optimisation.

7.2 On the Influence of Metrics

Analysing whether two metrics, i.e. objectives, are highly competitive is an important factor to properly face the multi-objective optimisation, since it might determine much of the complexity of problem to be solved. Regarding the problem under study, this could lead to situations in which architects might presume beforehand that some specific metrics are closely related, mostly because they apparently arrange the architectural design in a similar way.

Table 9 presents the range of objective values of the final solutions considering all the bi-objective problems. The symbol ▲ indicates that the metric values should be maximised, whereas ▼ stands for minimisation. A unique reference PF has been constructed per problem instance, which is composed of all the solutions that are still not dominated by any other found after 30 executions of all the algorithms for a given problem instance. Then, the minimum and maximum bounds have been calculated in order to determine the extent of each metric range. A value $x_{i,j}$ represents the range of values achieved by final solutions when a measure j is optimised jointly with the measure i. For example, looking at the joint optimisation of icd and erp ($x_{2,1}$), it can be observed that icd varies in the range [0.0, 0.7] when combined with erp, whereas the range of values obtained would vary from 0.4 to 0.8 if optimised together with the metric ins ($x_{3,1}$). Consequently, ins is less opposite to icd than erp, since higher values of icd can be achieved even when ins is highly optimised.

Notice that combinations like $icd - erp$, $cb - gcr$ or $enc - abs$ have a wide range of values for both measures. Therefore, when a solution reaches a near-optimal value for one measure, the obtained value for the other measure is very poor. In contrast, there are other combinations in which one measure has been highly optimised regardless of the other measure being considered, such as when cl is chosen. The type of problem instance and the possibility of creating architectural solutions with different number of components allow not exceeding the configured critical link threshold. In such a case, a search process should be capable of finding solutions that reach good values for both objectives at the same time, making the MOP resolution slightly simpler. This would also explain why some algorithms are tied (see Section 6.1).

The choice of design metrics also guides the search for certain types of architectural specifications. For example, a complex scenario is developed regarding the modularity criteria, where a trade-off between coupling and cohesion would be expected. In this sense, metrics like icd or ins imply searching for solutions comprising of more components as a way to

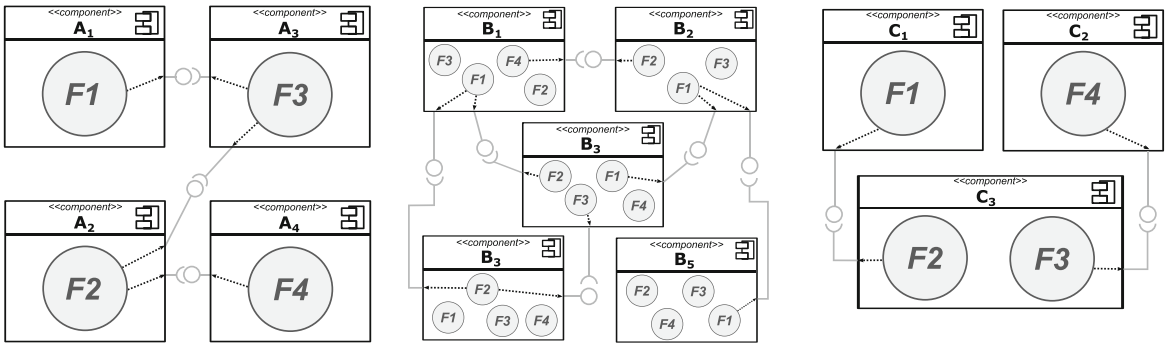
Table 9 Range of absolute objective values for non-dominated solutions considering all the combinations of 2 measures

	$icd^{\blacktriangle}$	$erp^{\blacktriangledown}$	$ins^{\blacktriangledown}$	$enc^{\blacktriangle}$	$cs^{\blacktriangledown}$	$cl^{\blacktriangledown}$	$gcr^{\blacktriangledown}$	$abs^{\blacktriangle}$	$cb^{\blacktriangle}$
$icd^{\blacktriangle}$	–	[0.0,178.0]	[0.1,0.6]	[0.9,1.0]	[0.0,0.2]	[0.0,0.0]	[1.0,1.4]	[0.1,0.9]	[0.3,1.0]
$erp^{\blacktriangledown}$	[0.0,0.7]	–	[0.1,0.5]	[0.5,1.0]	[0.0,0.3]	[0.0,0.0]	[1.0,1.7]	[0.1,0.9]	[0.1,1.0]
$ins^{\blacktriangledown}$	[0.4,0.8]	[0.0,84.0]	–	[0.7,1.0]	[0.0,0.1]	[0.0,0.0]	[1.0,8.3]	[0.3,0.9]	[0.1,1.0]
$enc^{\blacktriangle}$	[0.5,0.7]	[0.0,161.0]	[0.1,0.5]	–	[0.0,1.0]	[0.0,0.0]	[1.0,4.0]	[0.1,0.9]	[0.0,1.0]
$cs^{\blacktriangledown}$	[0.3,0.8]	[0.0,258.0]	[0.1,0.2]	[0.6,1.0]	–	[0.0,0.0]	[1.0,7.4]	[0.5,0.9]	[0.1,1.0]
$cl^{\blacktriangledown}$	[0.6,0.8]	[0.0,0.0]	[0.1,0.2]	[0.9,1.0]	[0.0,0.1]	–	[1.0,1.2]	[0.8,0.9]	[0.6,1.0]
$gcr^{\blacktriangledown}$	[0.3,0.7]	[0.0,6.0]	[0.1,0.6]	[0.5,1.0]	[0.0,0.2]	[0.0,0.0]	–	[0.3,0.9]	[0.0,1.0]
$abs^{\blacktriangle}$	[0.0,0.7]	[0.0,34.0]	[0.1,0.7]	[0.2,1.0]	[0.0,0.2]	[0.0,0.0]	[1.0,3.70]	–	[0.1,1.0]
$cb^{\blacktriangle}$	[0.2,0.7]	[0.0,560.0]	[0.1,0.5]	[0.4,1.0]	[0.0,0.2]	[0.0,1.0]	[1.0,22.0]	[0.1,0.9]	–

improve their cohesion. If the analysability is considered, then the architectures obtained would be composed of smaller components, each providing a well defined set of services. On the contrary, *erp* or *enc* look for a low coupling, so the solution structure is characterised by the creation of architectures with only two or three large components. Here, minimising the interactions among software artefacts would benefit the overall security and performance of the resulting system, although an excessive encapsulation of functionalities could harm its reusability and analysability, as observed when *abs* or *cb* were considered. Metrics like *cs* or *cb* counteract this effect and propitiate the generation of solutions comprised of a larger number of components.

The trade-off among these design criteria becomes a great challenge when 8 or 9 objectives have to be simultaneously optimised. In this case, low and medium values have been obtained for *icd*, especially if SPEA2, GrEA, IBEA and NSGA-III are executed. Similarly, *cb* and *abs* are difficult to optimise. IBEA can only provide solutions comprising components with unbalanced sizes even when *cs* is considered. Focusing on *abs*, it is worth mentioning that only NSGA-II, ϵ-MOEA and HypE reach values good enough for all the problem instances. The *cs* measure is usually cancelled out due to the presence of others like *erp*, *gcr* or *enc*. In this sense, the optimisation of these metrics seems to be harder, which makes the evolutionary process oriented towards them. As a result, these architectural solutions sometimes contain at least one large component. It should be noted that including *erp*, *gcr* and *abs* within the set of objectives leads to more variable outcomes over the different problem instances. In fact, these metrics are based on specific analysis information extracted from the original design, such as the number of abstract classes or the number and types of interrelationships. Finally, similar results are obtained for *ins* and *enc* in all the problem instances, meaning that these metrics are less conflicting.

To illustrate how the combination of metrics can influence the quality of the architectural description from the architectural perspective, the solutions returned by NSGA-II for the *Datapro4j* problem instance are scrutinised and discussed. Figure 8a depicts the original system architecture, which is comprised of 4 components ($A_1, \ldots, A_4$) whose functionalities will be referred here from $F1$ to $F4$. Notice that, according to the problem statement, the functionality of a component is represented as a group of highly interconnected classes, whose relationships are originally specified in the input class diagram. If *cs* and *cb* are considered together, one of the most interesting solutions returned by the algorithm is formed by 5 components ($B_1, \ldots, B_5$) with similar sizes (see Fig. 8b). The evaluation mechanism classifies all the components as noncritical in terms of their sizes, and both objectives are highly optimised. However, the set of classes inside each component does not represent a comprehensive functionality, but a combination of unrelated classes. The reason for this is that the optimisation process is directed towards the optimal size balance without considering



(a) Sketched original architecture (b) Sketched solution for $cs - cb$ (c) Sketched solution for $icd - erp$

Fig. 8 Returned solutions by NSGA-II for *Datapro4j* problem instance

whether the classes allocated in the same component are related. As a result, functionalities are distributed among different components, decreasing their cohesion and increasing the overall coupling. On the contrary, when icd and erp are the considered metrics, the algorithm found one solution with 3 components very similar to the original proposal (see Fig. 8c). Only the third component, C_3, comprised the classes related to $F2$ and $F3$, being catalogued as critical according to the cs metric. Finally, the joint optimisation of the 4 metrics leads to the discovery of solutions very close or even equal to the original architecture. Considering icd and erp as objective functions has served to properly identify the different functionalities, while cs and cb have encouraged the creation of two medium-sized components, one that contains the classes associated with $F2$ and another specifying $F3$. It should be noted that other problem instances could require different metrics, or even a greater number of objectives, in order to discover the most appropriate system architecture to the engineer. Nevertheless, in the cases in which the software architect is not confident enough with the intended architecture, considering the execution of different combinations of metrics could be appropriate. In short, the selection of metrics made by the architect is a key factor to determine the structure of the returned architectural solutions, whereas the achieved level of optimisation for the selected objectives is more related to the used algorithm.

7.3 On the Selection of the Evolutionary Algorithm

The selection of the evolutionary approach is not a trivial task for the user, not least if he needs to be aware of the impact that the diverse characteristics of each algorithm may have on searching for the expected architectonical solutions. SPEA2 shows a low performance in its search for high quality solutions, even though its diversity preservation technique has shown to be a very effective mechanism to provide a great variety of architectural solutions. On the other hand, NSGA-II provides a valuable scalability with respect to all the considered optimisation problems. Besides, regarding the diversity of solutions, this algorithm behaves more variably. Nevertheless, its scalable execution time and the absence of parameters are appealing features to select this algorithm as a generic choice.

The decomposition approach proposed by MOEA/D is an interesting characteristic to promote the diversity of solutions, but it works better with fewer than 6 objectives. Besides, the exploration of many search directions could lead to an excessive number of returned solutions, their diversity not always being guaranteed. These circumstances, along with the observed difficulties in generating high quality solutions, imply that the expert faces a complex task when looking for the final solution among all the alternatives given by the algorithm. Similarly, NSGA-III looks for the diversity of solutions, performing better than MOEA/D in highly dimensional objective spaces. In addition, this many-objective approach is capable of keeping most of the beneficial characteristics of its predecessor.

Concerning the rest of algorithms and considering the different families, it can be noted that GrEA is overtaken by ϵ-MOEA, whereas HypE shows a better behaviour than IBEA. One observed weakness of GrEA is that it suffers from an excessive convergence and also has some difficulty in removing invalid solutions during the search process. As a result, this algorithm seems to be unable to discover any competitive software architecture for the most complex systems. Something similar happens when ϵ-MOEA is executed over certain combinations of measures. Then, the algorithm could be more sensible to the selection of the ϵ values, whose configuration might be a laborious task for the expert. Nevertheless, ϵ-MOEA provides the best trade-off between convergence and diversity of solutions for a large number of metrics.

In general, IBEA is unable to reach a reasonable trade-off between the set of metrics, whose cardinality has an impact on its behaviour respecting the diversity of solutions. On the contrary, HypE, whose performance improves as the number of objectives increases, is able to find non-dominated solutions for the totality of the considered problems, showing an appropriate explorative capability. The only drawback of HypE, which is also observed in ϵ-MOEA, is its highly computational cost, especially if more than 6 metrics participate in the optimisation problem.

A better knowledge about the specific behaviour of each algorithm can determine its applicability to a particular design scenario. Even so, some general guidelines for the selection of the evolutionary approach are inferred from this study. In case the architect is requested to consider a specific set of metrics, the results in Section 6 would allow him to identify the best algorithm for that specific combination. Additionally, if a general choice is preferred, NSGA-II and ϵ-MOEA have demonstrated to have the best performance in general terms, so the final decision should be made considering other additional factors like the total number of metrics to be optimised or time requirements.

8 Threats to Validity

In order to precisely designate the design concepts and terms that lay the foundation for the proposed approach, the ISO Std. 25000 has been followed as a reference to correctly define the non-functional requirements that guide the search process. Additionally, the OMG standard UML 2 has served to provide a rigorous notation for the specification of the underlying design models at all the abstraction levels, as well as to ensure the semantic validity of quality metrics. Using a standard proposal like UML 2 brings the optimisation approach closer to the everyday realm of the software architect.

Internal validity refers to those aspects of the experimentation that cannot ensure the causality between the hypothesis and the obtained results. In SBSE, these aspects are related to the algorithm's set-up and its parametrisation. The use of default parameter values could produce bias on the results, since the original algorithms were usually tested over real optimisation problems, even though the nature of our problem is quite different. Here, the guidelines provided by their corresponding authors have been followed to configure the algorithms. Nevertheless, some preliminary experiments were performed to check their suitability. If required, some modifications to the original proposal implementations were made to adapt them to our specific combinatorial problem, e.g. the required constraint handling techniques were included in all the algorithms. These changes can affect the performance of the algorithms, but they are necessary to deal with the defined search problem and would affect all the algorithms in a similar way. The experimentation has been designed to avoid any bias due to the intrinsic randomness of the evolutionary approaches. So, every algorithm has been executed 30 times. Additionally, a hardware platform specifically dedicated to experimentation purposes has served to assure that all the algorithms have been executed under the same conditions, meaning that their execution time is also comparable. Additionally, well-known quality indicators like hypervolume and spacing have been used to evaluate the evolutionary performance of the algorithms. Non-parametric statistical tests at a significance level of 95 % were executed to draw precise conclusions in the light of the comparative results of the algorithms under study.

External validity is related with the generalisation of the experimental results. On the one hand, a set of real-world software systems has been considered as problem instances, covering different sizes and complexities. In this paper, the evolutionary search has been performed without any prior knowledge about the original design. It is worth mentioning that any additional analysis information would help to obtain more accurate outcomes from

the discovery process. On the other hand, the application of this work to other industrial domains might imply adapting the proposed approach to their specific requirements. The selected design metrics are closely related to maintainability, which is clearly an important characteristic in the design of component-based software architectures. Nevertheless, notice that addressing different design tasks based on other architectural models (e.g. service-oriented architectures, grid-based platforms, etc.) would probably require the evaluation of specific quality criteria.

9 Concluding Remarks

This paper presents a comparative study on the performance of different multi- and many-objective evolutionary algorithms for software architecture discovery, an abstract problem that demands diverse decisions to be made according to a number of requirements. The proposed experimental framework explores the ability of these algorithms to simultaneously deal with different metrics related to maintainability at the architectural level, and put forth some ideas about how they should selected and combined by the software architect. Existing dependencies among metrics have shown to be an important factor that could subsequently impact on both the complexity of the optimisation problem to be solved and the type of the architectural solutions returned by the search process. On the one hand, metrics like icd, ins, erp or gcr are more related to the allocation of classes within components on the basis of their relationships. On the other hand, aspects like avoiding critical components or balancing their sizes might be considered as secondary goals, since they are not capable of identifying the system functionalities.

The results obtained have shown that many-objective evolutionary algorithms provide an interesting alternative to deal with such a complex combinatorial problem. On the one hand, NSGA-II is the only multi-objective approach that achieves competitive results for some objective combinations with respect to many-objective approaches like ϵ-MOEA and HypE. Nevertheless, these algorithms provide a greater performance in terms of the expected trade-off between convergence and diversity when highly dimensional objective spaces need to be considered, as it is often the case of software design optimisation. On the other hand, the set of metrics selected to guide the search becomes a less influential factor on the performance of the algorithms as the number of objectives increases.

An in-depth analysis of the strengths of each algorithm has been presented and extensively discussed. The number of returned solutions or the execution time required to perform the search constitute complementary and valuable aspects to be considered in order to take a justified decision about the selection of a specific algorithm. Additionally, the guidelines provided in this study are essential to take a step further in the automatic recommendation of the evolutionary approach that best fits into a particular architectural problem. As a future work, hyper-heuristic approaches based on the information extracted from this analysis will serve as a basis for the development of decision support tools for software engineers. Finally, many-objective evolutionary optimisation is an open field of research, which makes it interesting to continue exploring some emerging approaches (He et al. 2014; Wang et al. 2014).

Acknowledgments Work supported by the Spanish Ministry of Science and Technology, project TIN2011-22408, the Spanish Ministry of Economy and Competitiveness, project TIN2014-55252-P, and FEDER funds. This research was also supported by the Spanish Ministry of Education under the FPU program (FPU13/01466).

Table 10 Results of the Cliff's Delta test for hypervolume (n=negligible, s=small, m=medium, l=large) ($\alpha = 0.05$)

Algorithm	SPEA2	NSGA-II	MOEA/D	ϵ-MOEA	GrEA	IBEA	HypE	NSGA-III
SPEA2	–	−1.00 (l)	−1.00 (l)	−1.00 (l)	−0.60 (l)	0.90 (l)	−1.00 (l)	−0.54 (l)
NSGA-II	0.90 (l)	–	0.14 (n)	−0.88 (l)	0.84 (l)	0.90 (l)	−0.06 (n)	0.90 (l)
MOEA/D	0.90 (l)	−0.14 (n)	–	−0.88 (l)	0.78 (l)	0.90 (l)	−0.20 (s)	0.90 (l)
ϵ-MOEA	0.90 (l)	0.88 (l)	0.88 (l)	–	0.90 (l)	0.90 (l)	0.88 (l)	0.90 (l)
GrEA	0.60 (l)	−0.90 (l)	−0.84 (l)	−1.00 (l)	–	0.60 (l)	−0.96 (l)	0.60 (s)
IBEA	−1.00 (l)	−1.00 (l)	−1.00 (l)	−1.00 (l)	−0.60 (l)	–	−1.00 (l)	−1.00 (l)
HypE	0.90 (l)	0.06 (n)	−0.20 (s)	−0.88 (l)	0.88 (l)	0.90 (l)	–	0.90 (l)
NSGA-III	0.54 (l)	−1.00 (l)	−1.00 (l)	−1.00 (l)	−0.60 (l)	0.900 (l)	−1.00 (l)	–

Appendix

Tables 10 and 11 present the results of the Cliff's Delta test for hypervolume (HV) and spacing (S), respectively. The value of a cell, $x_{i,j}$, represents the effect size and its interpretation when comparing the algorithm i against the algorithm j for the 9-objective optimisation problem in terms of the specific quality indicator. Table 12 shows the results of the Friedman and the Holm tests for all possible combinations of 2 objectives. Tables 13, 14, 15, 16 report the results for the 4-objective problems, whereas the 6-objective problems are shown from Table 17, 18, 19. In addition, Table 19 contains the results obtained from 8-objective and 9-objective problems. For each of them, the best rankings for the two quality indicators, HV and S, are shown in bold typeface, and their cells are shaded in gray colour when significant differences exist. The critical value, according to the F-Distribution with 6 and 54 degrees of freedom, i.e. the p-value, is 2.2720. Since the Holm test is performed if H_0 is rejected, Tables 12–19 show these ranking values in italic typeface when the corresponding algorithm lies below its critical threshold, i.e. its performance according to the column-specific indicator is worse than that provided by the best algorithm.

Table 11 Results of the Cliff's Delta test for spacing (n=negligible, s=small, m=medium, l=large) ($\alpha = 0.05$)

Algorithm	SPEA2	NSGA-II	MOEA/D	ϵ-MOEA	GrEA	IBEA	HypE	NSGA-III
SPEA2	–	0.32 (s)	0.89 (l)	0.87 (l)	0.87 (l)	0.90 (l)	0.85 (l)	0.68 (l)
NSGA-II	−0.32 (s)	–	0.90 (l)	0.90 (l)	0.88 (l)	0.90 (l)	0.87 (l)	0.88 (l)
MOEA/D	−0.98 (l)	−1.00 (l)	–	−0.36 (m)	0.64 (l)	0.90 (l)	−0.10 (n)	−0.88 (l)
ϵ-MOEA	−0.96 (l)	−1.00 (l)	0.35 (m)	–	0.78 (l)	0.90 (l)	0.26 (s)	−0.82 (l)
GrEA	−0.94 (l)	−0.96 (l)	−0.64 (l)	−0.78 (l)	–	0.58 (l)	−0.76 (l)	−0.82 (l)
IBEA	−1.00 (l)	−1.00 (l)	−1.00 (l)	−1.00 (l)	−0.58 (l)	–	−1.00 (l)	−1.00 (l)
HypE	−0.92 (l)	−0.94 (l)	0.10 (n)	−0.26 (s)	0.75 (l)	0.90 (l)	–	−0.64 (l)
NSGA-III	−0.68 (l)	−0.96 (l)	0.79 (l)	0.82 (l)	0.81 (l)	0.90 (l)	0.64 (l)	–

Table 12 Average rankings for 2-objective problems obtained from the Friedman test ($\alpha = 0.05$)

Objectives	SPEA2		NSGA-II		MOEA/D		ϵ-MOEA		GrEA		IBEA		HypE		NSGA-III	
	HV	S	HV	S	HV	S	HV	S	HV	S	HV	S	HV	S	HV	S
icd-erp	2.80	3.20	**1.30**	4.40	*5.35*	**2.20**	3.35	3.80	*4.70*	*5.80*	*7.75*	*7.50*	*4.10*	4.90	*6.65*	4.20
icd-gcr	4.00	**2.00**	**1.60**	4.20	*5.20*	2.80	3.50	*6.10*	3.80	*5.80*	*7.60*	*7.10*	3.60	3.50	*6.70*	4.50
icd-cs	*5.10*	3.65	2.45	2.60	3.85	**2.50**	4.70	*7.20*	3.85	5.05	*7.10*	*5.30*	**2.05**	4.00	*6.90*	*5.70*
icd-cl	*5.45*	*5.20*	**2.30**	*5.20*	3.90	4.60	2.50	5.20	3.90	*5.20*	*7.05*	**2.60**	3.35	*5.20*	*7.55*	2.80
icd-ins	*5.15*	3.10	2.90	3.70	*4.95*	**3.00**	**1.90**	*6.90*	4.25	5.00	*6.60*	3.10	2.50	4.20	*7.75*	*7.00*
icd-enc	*5.60*	2.40	**1.70**	4.30	3.80	**1.90**	2.80	*7.25*	4.20	*5.55*	*7.20*	*7.10*	3.30	3.30	*7.40*	4.20
icd-abs	2.70	3.20	**1.10**	*5.20*	*4.80*	**2.30**	*5.30*	3.95	*4.70*	*6.30*	*7.25*	*7.55*	2.80	4.60	*7.35*	2.90
icd-cb	3.90	**1.70**	2.30	3.80	*4.70*	1.80	*5.80*	*7.30*	3.50	*5.90*	*7.10*	4.10	**1.60**	4.20	*7.10*	*7.20*
erp-gcr	3.75	*5.20*	**2.75**	*5.80*	4.05	*4.60*	2.90	*5.90*	5.30	*5.30*	*7.30*	2.30	3.25	*5.40*	*6.70*	**1.50**
erp-cs	4.00	**2.80**	**1.70**	4.15	*4.40*	3.60	4.20	*6.75*	*6.25*	5.45	*7.40*	*5.55*	2.25	4.20	*5.80*	3.50
erp-cl	**3.20**	4.80	**3.20**	4.80	**3.20**	4.80	**3.20**	4.80	5.60	4.80	*7.55*	4.35	3.20	4.80	*6.85*	**2.85**
erp-ins	3.70	**2.20**	**2.25**	4.45	4.45	3.85	2.85	*5.65*	*5.10*	4.85	*7.65*	*5.90*	3.25	4.25	*6.75*	*4.85*
erp-enc	*3.60*	**2.50**	**1.35**	3.50	*3.80*	2.60	*3.80*	*6.20*	*6.20*	*6.10*	*7.35*	*6.15*	*3.60*	4.75	*6.30*	4.20
erp-abs	3.60	4.90	**1.60**	**3.30**	*4.40*	4.25	3.15	4.80	*5.90*	4.60	*7.60*	6.00	3.25	3.65	*6.50*	4.50
erp-cb	*4.20*	2.70	**1.50**	3.90	*4.60*	**2.20**	*4.40*	4.20	3.85	*5.55*	*7.45*	*7.35*	3.20	4.20	*6.80*	*5.90*
gcr-cs	4.20	**2.95**	**1.80**	4.25	3.95	3.35	3.85	*6.05*	*6.20*	5.25	*6.80*	4.85	2.45	3.65	*6.75*	5.65
gcr-cl	3.70	4.90	**3.20**	4.90	**3.20**	4.90	**3.20**	4.90	5.50	4.90	*7.15*	3.80	**3.20**	4.90	*6.85*	**2.80**
gcr-ins	4.70	**1.90**	**2.40**	4.20	4.80	3.70	2.80	*5.75*	4.20	*5.85*	*7.70*	*6.55*	2.80	4.35	*6.60*	3.70
gcr-enc	*4.60*	3.60	**1.30**	3.90	*3.95*	3.80	*3.95*	4.40	*5.70*	*6.50*	*7.00*	*6.40*	*3.50*	**3.20**	*6.00*	4.20
gcr-abs	4.95	5.00	**2.20**	**3.00**	4.70	3.50	2.95	5.20	4.40	4.85	*7.40*	*6.00*	3.00	4.45	*6.40*	4.00
gcr-cb	4.70	**2.60**	**2.25**	4.10	3.35	3.70	3.90	3.40	4.05	5.35	*7.25*	*7.05*	3.20	2.70	*7.30*	*7.10*
cs-cl	3.50	4.85	**2.55**	4.85	3.15	4.15	4.70	4.85	5.05	4.85	*7.05*	4.40	3.45	4.85	*6.55*	**3.20**
cs-ins	*4.45*	3.90	**1.65**	4.40	3.60	4.05	*5.75*	*6.00*	*5.95*	*5.70*	*6.65*	*5.90*	2.05	3.50	*5.90*	**2.55**

Table 12 (continued)

Objectives	SPEA2		NSGA-II		MOEA/D		ϵ-MOEA		GrEA		IBEA		HypE		NSGA-III	
	HV	S	HV	S	HV	S	HV	S	HV	S	HV	S	HV	S	HV	S
cs-enc	*4.80*	3.30	**1.70**	3.60	3.60	**2.05**	*5.10*	*6.75*	*5.35*	*5.10*	*6.95*	*6.50*	2.80	4.00	*5.70*	4.70
cs-abs	*4.35*	*5.10*	**1.80**	3.10	4.10	**1.70**	*5.25*	*7.25*	4.30	*4.75*	*7.40*	*6.90*	2.00	3.10	*6.80*	4.10
cs-cb	*5.05*	5.30	**2.05**	**3.15**	2.80	3.25	*5.50*	5.30	4.55	*5.30*	*6.70*	5.30	2.85	3.45	*6.50*	4.95
cl-ins	3.55	4.75	2.60	5.25	**2.55**	4.80	5.10	5.25	*5.15*	4.85	*7.40*	**2.65**	2.60	4.70	*7.05*	3.75
cl-enc	*6.05*	*5.05*	2.55	*5.05*	**2.40**	*5.05*	*5.55*	*5.05*	4.70	*5.05*	*6.35*	*4.25*	2.60	*5.05*	*5.80*	**1.45**
cl-abs	*5.25*	*5.15*	**1.70**	4.15	*4.65*	*5.15*	2.70	5.15	3.85	4.45	*7.35*	*5.15*	3.40	4.35	*7.10*	**2.45**
cl-cb	4.75	*5.80*	**2.60**	2.75	3.40	**2.35**	3.45	*5.80*	4.70	*5.15*	*7.10*	*5.80*	3.30	3.90	*6.70*	4.45
ins-enc	4.50	2.50	**2.45**	2.70	3.75	**2.50**	3.20	*6.25*	5.20	*5.95*	*7.15*	*6.70*	2.70	3.30	*7.05*	*6.10*
ins-abs	*4.80*	2.80	2.70	3.40	*5.20*	**1.40**	**1.30**	*5.75*	*4.20*	*5.30*	*7.00*	*7.35*	3.40	4.00	*7.40*	*6.00*
ins-cb	3.40	**2.20**	**1.75**	3.40	3.75	2.30	3.70	*5.35*	*6.10*	*6.30*	*7.35*	*7.45*	3.10	3.30	*6.85*	*5.70*
enc-abs	*4.45*	**1.90**	**1.10**	3.10	2.30	3.70	*4.80*	*5.85*	*5.10*	*5.30*	*7.20*	*7.35*	*4.20*	3.80	*6.85*	*5.00*
enc-cb	*5.30*	*5.20*	**1.40**	2.80	2.80	**1.40**	*4.90*	*6.85*	*4.35*	*5.10*	*7.20*	*6.60*	3.60	2.40	*6.45*	*5.65*
abs-cb	*4.80*	3.00	**1.10**	4.70	2.90	**2.40**	*3.60*	3.80	*5.25*	*6.30*	*7.40*	*7.45*	*3.80*	5.10	*7.15*	3.25
cl-cb	4.75	*5.80*	**2.60**	2.75	3.40	**2.35**	3.45	*5.80*	4.70	*5.15*	*7.10*	*5.80*	3.30	3.90	*6.70*	4.45
ins-enc	4.50	2.50	**2.45**	2.70	3.75	**2.50**	3.20	*6.25*	5.20	*5.95*	*7.15*	*6.70*	2.70	3.30	*7.05*	*6.10*
ins-abs	*4.80*	2.80	2.70	3.40	*5.20*	**1.40**	**1.30**	*5.75*	*4.20*	*5.30*	*7.00*	*7.35*	3.40	4.00	*7.40*	*6.00*
ins-cb	3.40	**2.20**	**1.75**	3.40	3.75	2.30	3.70	*5.35*	*6.10*	*6.30*	*7.35*	*7.45*	3.10	3.30	*6.85*	*5.70*
enc-abs	*4.45*	**1.90**	**1.10**	3.10	2.30	3.70	*4.80*	*5.85*	*5.10*	*5.30*	*7.20*	*7.35*	*4.20*	3.80	*6.85*	*5.00*
enc-cb	*5.30*	*5.20*	**1.40**	2.80	2.80	**1.40**	*4.90*	*6.85*	*4.35*	*5.10*	*7.20*	*6.60*	3.60	2.40	*6.45*	*5.65*
abs-cb	*4.80*	3.00	**1.10**	4.70	2.90	**2.40**	*3.60*	3.80	*5.25*	*6.30*	*7.40*	*7.45*	*3.80*	5.10	*7.15*	3.25

Table 13 Average rankings for 4-objective problems obtained from the Friedman Test ($\alpha = 0.05$) (cont'd)

Objectives	SPEA2		NSGA-II		MOEA/D		ϵ-MOEA		GrEA		IBEA		HypE		NSGA-III	
	HV	S	HV	S	HV	S	HV	S	HV	S	HV	S	HV	S	HV	S
icd-erp-gcr-cs	2.10	3.00	**1.00**	*6.00*	*4.20*	3.40	*4.00*	**2.40**	*5.75*	*6.30*	*7.75*	*7.40*	*4.60*	5.00	*6.60*	2.50
icd-erp-gcr-cl	2.40	2.90	**1.40**	*5.60*	*4.80*	3.20	3.50	3.10	*5.75*	*6.70*	*7.65*	*7.30*	3.90	4.70	*6.60*	**2.50**
icd-erp-gcr-ins	2.90	**2.10**	**1.30**	*5.80*	*4.50*	2.70	2.00	2.40	*6.20*	*6.90*	*7.60*	*7.30*	*4.90*	4.70	*6.60*	4.10
icd-erp-gcr-enc	3.20	2.90	**1.10**	*5.80*	*4.10*	2.80	2.30	**1.50**	*6.40*	*7.10*	*7.60*	*7.50*	*4.70*	*4.60*	*6.60*	3.80
icd-erp-gcr-abs	*3.70*	**1.30**	**1.10**	*5.20*	*4.10*	3.00	1.90	2.70	*6.70*	*7.20*	*7.60*	*7.60*	*4.30*	*4.20*	*6.60*	*4.80*
icd-erp-gcr-cb	*4.40*	1.90	**1.10**	*5.20*	*4.80*	3.00	1.90	**1.90**	*5.10*	*6.80*	*7.80*	*7.60*	*4.30*	*5.00*	*6.60*	*4.60*
icd-erp-cs-cl	2.40	2.70	**1.20**	*5.70*	*4.20*	**2.50**	*5.30*	4.65	*5.50*	*6.45*	*7.70*	*7.20*	3.10	3.80	*6.60*	3.00
icd-erp-cs-ins	*3.60*	2.20	**1.00**	*5.10*	*5.10*	2.50	2.80	**1.90**	*5.00*	*7.20*	*7.70*	*7.60*	*4.20*	*5.20*	*6.60*	4.30
icd-erp-cs-enc	*5.40*	**1.40**	**1.00**	*4.70*	2.70	*4.30*	*4.40*	2.15	*5.10*	*6.85*	*7.70*	*7.50*	3.10	*5.30*	*6.60*	3.80
icd-erp-cs-abs	*5.60*	**1.00**	**1.10**	*3.60*	*4.10*	*3.80*	1.90	2.20	*5.70*	*7.20*	*7.50*	*7.50*	3.40	*5.60*	*6.70*	*5.10*
icd-erp-cs-cb	*5.30*	**1.20**	**1.60**	*5.00*	*4.90*	3.30	1.90	2.10	*4.45*	*6.70*	*7.75*	*7.60*	3.40	*6.10*	*6.70*	*4.00*
icd-erp-cl-ins	3.35	2.60	**1.50**	*5.60*	*5.10*	3.50	2.45	**1.80**	*5.20*	*6.10*	*7.75*	*7.80*	4.00	4.00	*6.65*	*4.60*
icd-erp-cl-enc	*3.50*	2.60	**1.00**	*6.10*	*4.30*	3.10	3.00	**2.30**	*6.35*	*7.10*	*7.65*	*7.40*	*3.60*	4.80	*6.60*	2.60
icd-erp-cl-abs	*4.00*	**1.30**	**1.00**	*5.30*	*4.60*	3.70	2.00	1.80	*6.45*	*7.40*	*7.75*	*7.60*	*3.60*	*4.20*	*6.60*	*4.70*
icd-erp-cl-cb	*5.20*	**1.70**	**1.30**	*5.20*	*4.90*	2.90	2.20	2.00	*5.15*	*6.80*	*7.70*	*7.70*	2.80	*5.20*	*6.75*	*4.50*
icd-erp-ins-enc	*3.90*	2.00	**1.20**	*5.10*	*4.30*	3.60	1.90	**1.30**	*6.05*	*7.05*	*7.65*	*7.65*	*4.40*	*5.20*	*6.60*	*4.10*
icd-erp-ins-abs	*4.20*	**1.00**	**1.50**	3.30	*4.70*	*3.70*	1.50	2.50	*6.25*	*6.95*	*7.65*	*7.65*	3.60	*5.50*	*6.60*	*5.40*
icd-erp-ins-cb	*5.00*	**1.20**	**1.60**	*4.30*	*5.10*	3.60	1.60	1.80	*4.90*	*6.75*	*7.75*	*7.75*	3.40	*5.90*	*6.65*	*4.70*
icd-erp-enc-abs	*4.40*	**1.10**	**1.40**	*4.00*	3.70	3.40	1.60	3.00	*6.60*	*7.15*	*7.80*	*7.65*	3.90	*4.60*	*6.60*	*5.10*
icd-erp-enc-cb	*5.60*	**1.00**	**1.30**	*3.80*	*4.40*	*4.00*	1.70	2.00	*5.30*	*7.20*	*7.80*	*7.60*	3.30	*5.60*	*6.60*	*4.80*
icd-erp-abs-cb	*5.70*	**1.00**	1.60	2.80	*4.60*	*4.10*	**1.40**	2.50	*5.40*	*7.30*	*7.80*	*7.60*	3.00	*5.60*	*6.50*	*5.10*
icd-gcr-cs-cl	3.50	3.10	**1.40**	4.50	*5.00*	**2.30**	*4.90*	*6.00*	3.75	*6.00*	*7.65*	*7.40*	3.00	4.00	*6.80*	2.70

Table 13 (continued)

Objectives	SPEA2		NSGA-II		MOEA/D		ϵ-MOEA		GrEA		IBEA		HypE		NSGA-III	
	HV	S	HV	S	HV	S	HV	S	HV	S	HV	S	HV	S	HV	S
icd-gcr-cs-ins	3.90	2.10	**1.40**	*4.60*	*5.20*	3.20	3.40	**1.60**	*4.40*	*7.10*	*7.70*	*7.60*	3.40	*5.60*	*6.60*	4.20
icd-gcr-cs-enc	*5.00*	**1.50**	**1.30**	*4.80*	*3.90*	3.90	*4.50*	1.60	*4.55*	*7.20*	*7.75*	*7.60*	2.40	*5.30*	*6.60*	*4.10*
icd-gcr-cs-abs	*5.60*	**1.10**	**1.00**	*3.90*	*4.50*	3.10	2.00	2.10	*5.35*	*7.20*	*7.75*	*7.60*	3.20	*6.10*	*6.60*	*4.90*
icd-gcr-cs-cb	*5.40*	**1.20**	**2.20**	*4.90*	4.80	3.60	2.70	2.50	4.05	*6.35*	*7.65*	*7.65*	2.30	*4.50*	*6.90*	*5.30*
icd-gcr-cl-ins	*4.20*	2.10	**1.70**	*5.40*	*5.10*	2.80	2.60	**1.90**	4.00	*6.60*	*7.60*	*7.60*	4.10	*4.90*	*6.70*	*4.70*
icd-gcr-cl-enc	*4.50*	2.30	**1.10**	*6.20*	*4.30*	3.80	2.80	**1.80**	*5.10*	*7.30*	*7.60*	*7.40*	*4.00*	4.40	*6.60*	2.80
icd-gcr-cl-abs	*4.10*	**1.50**	**1.00**	*5.60*	*5.00*	3.20	2.00	2.40	*6.45*	*7.30*	*7.65*	*7.60*	3.20	3.40	*6.60*	*5.00*
icd-gcr-cl-cb	*4.70*	1.90	**1.40**	*5.00*	*5.40*	3.00	2.50	**1.50**	4.00	*6.80*	*7.55*	*7.60*	3.40	*4.50*	*7.05*	*5.70*
icd-gcr-ins-enc	*5.00*	**1.30**	**1.20**	*5.00*	*4.60*	*3.80*	2.00	1.80	*5.20*	*7.30*	*7.60*	*7.40*	*3.80*	*5.20*	*6.60*	*4.20*
icd-gcr-ins-abs	*4.50*	**1.10**	1.60	3.50	*5.30*	3.40	**1.40**	2.50	*5.90*	*7.40*	*7.60*	*7.60*	3.10	*5.60*	*6.60*	*4.90*

Table 14 Average rankings for 4-objective problems obtained from the Friedman Test ($\alpha = 0.05$) (cont'd)

Objectives	SPEA2		NSGA-II		MOEA/D		ϵ-MOEA		GrEA		IBEA		HypE		NSGA-III	
	HV	S	HV	S	HV	S	HV	S	HV	S	HV	S	HV	S	HV	S
icd-gcr-ins-cb	*4.80*	**1.20**	**1.50**	*4.60*	*5.40*	3.50	2.20	2.00	*4.65*	*6.90*	*7.75*	*7.60*	3.00	*5.80*	*6.70*	*4.40*
icd-gcr-enc-abs	*5.00*	**1.00**	1.60	*4.10*	3.80	*3.50*	**1.40**	*3.30*	*6.65*	*7.20*	*7.65*	*7.60*	3.30	*4.10*	*6.60*	*5.20*
icd-gcr-enc-cb	*5.60*	**1.20**	**1.20**	*4.90*	*4.60*	3.30	2.70	1.90	*4.90*	*7.10*	*7.80*	*7.60*	2.60	*5.40*	*6.60*	*4.60*
icd-gcr-abs-cb	*5.60*	**1.10**	1.60	3.30	*4.50*	3.50	**1.40**	2.30	*5.40*	*7.30*	*7.80*	*7.60*	3.10	*5.60*	*6.60*	*5.30*
icd-cs-cl-ins	5.00	3.50	**2.30**	*4.60*	*5.20*	**1.40**	2.70	*6.40*	3.60	*6.15*	*7.60*	*7.25*	2.60	*4.10*	*7.00*	2.60
icd-cs-cl-enc	*5.2*	**1.80**	**1.80**	4.40	4.00	1.90	4.50	*6.95*	3.80	*6.50*	*7.40*	*7.05*	2.40	4.00	*6.90*	3.40
icd-cs-cl-abs	*4.40*	3.00	**1.00**	*5.90*	*5.60*	3.10	*3.60*	**1.50**	*4.90*	*7.30*	*7.55*	*7.20*	2.00	*5.40*	*6.95*	2.60
icd-cs-cl-cb	*4.80*	2.90	**2.00**	*4.60*	*5.40*	**1.50**	2.70	*5.70*	3.90	*6.50*	*7.60*	*7.10*	2.60	4.10	*7.00*	3.60
icd-cs-ins-enc	*5.50*	**1.70**	2.20	*5.20*	*4.50*	3.10	3.20	2.10	4.45	*7.10*	*7.45*	*7.60*	**1.90**	*4.80*	*6.80*	*4.40*
icd-cs-ins-abs	*5.00*	**1.60**	1.60	*4.70*	*5.20*	4.20	**1.50**	2.10	*5.25*	*6.75*	*7.50*	*7.45*	2.90	*6.70*	*7.05*	2.50
icd-cs-ins-cb	*4.10*	3.10	2.60	*5.30*	*5.20*	**2.30**	*4.75*	4.20	*4.05*	*6.90*	*7.45*	*6.70*	**1.20**	4.60	*6.65*	2.90
icd-cs-enc-abs	*5.60*	**1.00**	**1.30**	*3.60*	*4.10*	*4.30*	2.40	2.00	*5.90*	*7.10*	*7.70*	*7.40*	2.30	*6.50*	*6.70*	*4.10*
icd-cs-enc-cb	*5.20*	**1.70**	2.40	*5.40*	*4.90*	3.10	2.80	2.10	4.35	*7.15*	*7.75*	*7.65*	**1.90**	*4.50*	*6.70*	*4.40*
icd-cs-abs-cb	*5.60*	**1.40**	**1.10**	*4.50*	*4.50*	*4.50*	2.20	2.30	*5.35*	*6.85*	*7.50*	*7.75*	2.70	*6.40*	*7.05*	2.30
icd-cl-ins-enc	*5.50*	**1.90**	**2.00**	*5.50*	4.50	2.40	2.50	3.30	3.95	*7.20*	*7.60*	*7.70*	3.20	4.60	*6.75*	3.40
icd-cl-ins-abs	*4.30*	2.20	1.60	*6.00*	*5.40*	3.50	**1.40**	**1.90**	*5.30*	*7.40*	*7.65*	*7.60*	3.50	4.50	*6.85*	2.90
icd-cl-ins-cb	4.40	3.20	2.70	*5.40*	*5.60*	**1.70**	**2.20**	2.00	3.75	*6.75*	*7.65*	*7.45*	2.80	*4.70*	*6.90*	*4.80*
icd-cl-enc-abs	*4.50*	**1.70**	**1.30**	*5.80*	*4.20*	3.60	1.70	2.20	*6.40*	*7.40*	*7.55*	*7.50*	3.40	4.40	*6.95*	3.40
icd-cl-enc-cb	*4.90*	2.70	**2.00**	*5.80*	*5.10*	2.20	2.60	**2.10**	4.40	*6.95*	*7.75*	*7.65*	2.60	*5.00*	*6.65*	3.60
icd-cl-abs-cb	*5.40*	**1.60**	**1.30**	*4.90*	*4.70*	3.80	2.10	2.00	*5.30*	*7.25*	*7.40*	*7.65*	2.60	*5.60*	*7.20*	3.20
icd-ins-enc-abs	*5.10*	**1.20**	1.80	*3.90*	*4.60*	3.50	**1.20**	2.40	*5.85*	*7.15*	*7.70*	*7.65*	3.00	*6.20*	*6.75*	*4.00*
icd-ins-enc-cb	*5.40*	1.70	2.50	*5.00*	*4.80*	2.90	2.20	**1.60**	*4.85*	*7.30*	*7.75*	*7.60*	**1.90**	*4.80*	*6.60*	*5.10*

Table 14 (continued)

Objectives	SPEA2		NSGA-II		MOEA/D		ϵ-MOEA		GrEA		IBEA		HypE		NSGA-III	
	HV	S	HV	S	HV	S	HV	S	HV	S	HV	S	HV	S	HV	S
icd-ins-abs-cb	*5.60*	**1.10**	2.00	*3.70*	*4.60*	*4.10*	**1.20**	2.20	*5.35*	*6.70*	*7.45*	*7.70*	2.80	*6.60*	*7.00*	*3.90*
icd-enc-abs-cb	*5.60*	**1.20**	2.30	3.60	*4.40*	*3.80*	**1.60**	2.10	*5.55*	*7.10*	*7.70*	*7.40*	2.10	*6.30*	*6.75*	*4.50*
erp-gcr-cs-cl	3.15	4.35	**1.70**	4.35	*4.60*	2.75	*4.55*	*7.20*	*5.05*	5.25	*7.65*	5.15	2.80	4.25	*6.50*	**2.70**
erp-gcr-cs-ins	*4.25*	**2.40**	**1.45**	*5.30*	*4.85*	3.60	3.50	4.10	*5.65*	*6.40*	*7.55*	*6.50*	2.45	4.40	*6.30*	3.30
erp-gcr-cs-enc	*4.10*	**2.00**	**1.00**	*5.50*	*3.90*	3.60	*4.80*	4.10	*6.30*	*6.80*	*7.30*	*7.00*	2.50	4.10	*6.10*	2.90
erp-gcr-cs-abs	*3.50*	**2.10**	**1.00**	*5.00*	*4.20*	2.20	*3.50*	*5.30*	*5.95*	*6.30*	*7.65*	*7.20*	*3.70*	3.10	*6.50*	*4.80*
erp-gcr-cs-cb	*3.70*	2.30	**1.20**	*5.10*	*4.60*	**2.10**	2.80	4.00	*4.60*	*6.60*	*7.80*	*7.50*	*4.70*	*5.60*	*6.60*	2.80
erp-gcr-cl-ins	4.15	*4.40*	**2.25**	*4.75*	3.20	**1.70**	2.75	*4.85*	*5.60*	*6.25*	*7.60*	*6.20*	3.85	*4.25*	*6.60*	3.60
erp-gcr-cl-enc	*5.35*	*6.55*	**1.10**	*4.80*	3.30	1.80	*5.65*	*5.50*	*5.25*	*5.95*	*7.10*	*5.80*	2.25	3.80	*6.00*	**1.80**
erp-gcr-cl-abs	*4.10*	4.75	**1.35**	**2.70**	*4.00*	2.80	3.05	*5.80*	*6.05*	*5.70*	*7.55*	*6.20*	3.40	3.55	*6.50*	4.50
erp-gcr-cl-cb	3.40	3.00	**1.10**	4.90	*5.30*	2.90	3.50	**2.70**	*5.60*	*6.80*	*7.80*	*7.40*	2.70	5.40	*6.60*	2.90

Table 15 Average rankings for 4-objective problems obtained from the Friedman Test ($\alpha = 0.05$) (cont'd)

Objectives	SPEA2		NSGA-II		MOEA/D		ϵ-MOEA		GrEA		IBEA		HypE		NSGA-III	
	HV	S	HV	S	HV	S	HV	S	HV	S	HV	S	HV	S	HV	S
erp-gcr-ins-enc	3.50	**2.40**	**1.20**	*5.90*	*4.00*	3.80	2.00	2.70	*6.60*	*6.30*	*7.60*	*7.50*	*4.50*	3.90	*6.60*	3.50
erp-gcr-ins-abs	3.40	**1.90**	1.90	*4.90*	*4.40*	2.20	**1.30**	4.50	*6.30*	*5.70*	*7.60*	*7.60*	*4.50*	2.80	*6.60*	*6.40*
erp-gcr-ins-cb	*4.20*	**1.60**	**1.10**	*5.20*	*4.20*	3.10	2.30	2.40	*5.70*	*6.45*	*7.80*	*7.65*	*4.10*	*6.10*	*6.60*	3.50
erp-gcr-enc-abs	*4.50*	3.00	**1.00**	4.60	2.90	**3.00**	*3.70*	3.50	*6.30*	*6.60*	*7.40*	*7.00*	*3.80*	4.00	*6.40*	4.30
erp-gcr-enc-cb	*4.10*	**1.20**	**1.00**	*5.30*	*4.00*	3.60	2.40	1.90	*6.60*	*6.70*	*7.80*	*7.60*	*3.50*	*5.90*	*6.60*	*3.80*
erp-gcr-abs-cb	*4.60*	**1.10**	**1.10**	*4.40*	*4.70*	3.20	2.10	2.50	*5.80*	*7.20*	*7.80*	*7.60*	3.30	*5.10*	*6.60*	*4.90*
erp-cs-cl-ins	2.85	3.40	**1.55**	*5.10*	4.10	**1.60**	*4.25*	*6.00*	*5.50*	*5.85*	*7.60*	*6.85*	3.55	4.20	*6.60*	3.00
erp-cs-cl-enc	*5.35*	*6.55*	**1.10**	*4.80*	3.30	1.80	*5.65*	*5.50*	*5.25*	*5.95*	*7.10*	*5.80*	2.25	3.80	*6.00*	**1.80**
erp-cs-cl-abs	3.40	**2.10**	**1.00**	4.60	*4.50*	**2.10**	*4.90*	*5.15*	*5.35*	*6.15*	*7.55*	*7.30*	2.80	3.50	*6.50*	*5.10*
erp-cs-cl-cb	*4.70*	2.60	**1.50**	*5.10*	*4.70*	**1.60**	3.80	*4.30*	3.90	*6.80*	*7.80*	*7.70*	3.00	*5.10*	*6.60*	2.80
erp-cs-ins-enc	*4.95*	**1.60**	**1.10**	*4.70*	*4.75*	3.90	3.15	2.00	*5.85*	*6.90*	*7.30*	*7.60*	2.60	*5.40*	*6.30*	3.90
erp-cs-ins-abs	*4.80*	**1.50**	**1.10**	*4.30*	*4.40*	2.50	2.10	3.30	*5.20*	*7.10*	*7.60*	*7.60*	*4.20*	4.00	*6.60*	*5.70*
erp-cs-ins-cb	*5.50*	**1.70**	**1.40**	*5.10*	*4.60*	3.20	2.00	2.20	*4.70*	*6.70*	*7.80*	*7.70*	3.40	*6.50*	*6.60*	2.90
erp-cs-enc-abs	*5.40*	**1.00**	**1.00**	3.00	2.60	*3.90*	*4.00*	*3.70*	*5.90*	*6.40*	*7.60*	*7.40*	3.00	*5.80*	*6.50*	*4.80*
erp-cs-enc-cb	*5.40*	**1.30**	**1.00**	3.90	*4.00*	*4.00*	*3.60*	2.60	*4.60*	*7.10*	*7.80*	*7.60*	3.10	*6.30*	*6.50*	3.20
erp-cs-abs-cb	*5.30*	**1.30**	**1.10**	3.10	*4.50*	3.50	2.50	2.30	*5.50*	*7.15*	*7.80*	*7.65*	2.70	*6.10*	*6.60*	*4.90*
erp-cl-ins-enc	*3.60*	2.80	**1.10**	*6.00*	*3.60*	3.30	2.50	**1.20**	*6.35*	*7.15*	*7.65*	*7.55*	*4.60*	*4.00*	*6.60*	*4.00*
erp-cl-ins-abs	3.70	2.20	2.00	4.10	*4.40*	2.40	**1.50**	4.20	*5.65*	*6.90*	*7.60*	*7.60*	*4.50*	**2.20**	*6.65*	*6.40*
erp-cl-ins-cb	*4.90*	**1.60**	**1.30**	*5.20*	*4.90*	3.10	2.20	1.70	*5.50*	*6.60*	*7.70*	*7.80*	2.80	*5.00*	*6.70*	*5.00*
erp-cl-enc-abs	*4.60*	3.20	**1.00**	4.40	2.80	**2.90**	*4.75*	4.00	*5.95*	*6.50*	*7.60*	*7.10*	2.90	3.60	*6.40*	4.30
erp-cl-enc-cb	*5.10*	**1.10**	**1.00**	*5.10*	*4.20*	3.20	2.90	2.40	*5.95*	*7.00*	*7.75*	*7.60*	2.50	*4.70*	*6.60*	*4.90*
erp-cl-abs-cb	*4.90*	**1.20**	**1.10**	*4.50*	*4.80*	2.80	2.60	2.40	*5.90*	*6.95*	*7.80*	*7.65*	2.30	*4.80*	*6.60*	*5.70*
erp-ins-enc-abs	*4.60*	**1.30**	1.50	3.20	3.80	3.40	**1.50**	3.00	*6.35*	*7.10*	*7.75*	*7.60*	3.90	*5.40*	*6.60*	*5.00*

Table 15 (continued)

Objectives	SPEA2		NSGA-II		MOEA/D		ϵ-MOEA		GrEA		IBEA		HypE		NSGA-III	
	HV	S	HV	S	HV	S	HV	S	HV	S	HV	S	HV	S	HV	S
erp-ins-enc-cb	*5.60*	**1.10**	**1.50**	*3.60*	*4.40*	*4.30*	1.50	2.10	*5.10*	*7.10*	*7.80*	*7.60*	3.50	*5.70*	*6.60*	*4.50*
erp-ins-abs-cb	*5.60*	**1.00**	1.60	3.30	*4.40*	*3.70*	**1.50**	2.60	*5.30*	*7.30*	*7.80*	*7.60*	3.20	*5.70*	*6.60*	*4.80*
erp-enc-abs-cb	*5.30*	**1.00**	**1.50**	2.80	4.10	*4.40*	2.40	2.90	*5.90*	*6.70*	*7.75*	*7.60*	2.40	*5.20*	*6.65*	*5.40*
gcr-cs-cl-ins	2.75	2.90	**1.65**	*5.55*	*4.75*	**1.70**	*4.70*	*5.85*	*4.45*	*5.95*	*7.75*	*7.25*	3.35	4.20	*6.60*	2.60
gcr-cs-cl-enc	*4.95*	*4.60*	**1.00**	*4.60*	2.90	2.60	*6.85*	*6.90*	*4.90*	*6.00*	*7.00*	*6.30*	2.70	3.40	*5.70*	**1.60**
gcr-cs-cl-abs	*4.30*	3.25	**1.15**	*4.40*	*5.10*	**1.30**	*4.00*	*5.90*	*4.85*	*6.65*	*7.65*	*7.10*	2.35	3.10	*6.60*	*4.30*
gcr-cs-cl-cb	*4.90*	**2.10**	**1.50**	*5.00*	*5.00*	2.50	3.50	3.50	*4.15*	*5.95*	*7.60*	*7.65*	2.50	*4.70*	*6.85*	4.60
gcr-cs-ins-enc	*5.20*	1.70	**1.00**	*4.50*	*4.80*	*4.70*	*3.70*	**1.40**	*4.85*	*6.80*	*7.55*	*7.60*	2.60	*5.90*	*6.30*	3.40
gcr-cs-ins-abs	*4.70*	**1.30**	**1.30**	*4.50*	*5.20*	2.40	2.30	3.00	*5.05*	*6.90*	*7.75*	*7.60*	3.10	*4.50*	*6.60*	*5.80*
gcr-cs-ins-cb	*5.40*	**1.70**	**1.20**	*5.00*	*4.60*	4.30	2.40	2.10	*4.60*	*6.70*	*7.80*	*7.80*	3.40	*6.20*	*6.60*	2.20
gcr-cs-cl-ins	2.75	2.90	**1.65**	*5.55*	*4.75*	**1.70**	*4.70*	*5.85*	*4.45*	*5.95*	*7.75*	*7.25*	3.35	4.20	*6.60*	2.60
gcr-cs-cl-enc	*4.95*	*4.60*	**1.00**	*4.60*	2.90	2.60	*6.85*	*6.90*	*4.90*	*6.00*	*7.00*	*6.30*	2.70	3.40	*5.70*	**1.60**
gcr-cs-cl-abs	*4.30*	3.25	**1.15**	*4.40*	*5.10*	**1.30**	*4.00*	*5.90*	*4.85*	*6.65*	*7.65*	*7.10*	2.35	3.10	*6.60*	*4.30*
gcr-cs-cl-cb	*4.90*	**2.10**	**1.50**	*5.00*	*5.00*	2.50	3.50	3.50	*4.15*	*5.95*	*7.60*	*7.65*	2.50	*4.70*	*6.85*	4.60
gcr-cs-ins-enc	*5.20*	1.70	**1.00**	*4.50*	*4.80*	*4.70*	*3.70*	**1.40**	*4.85*	*6.80*	*7.55*	*7.60*	2.60	*5.90*	*6.30*	3.40
gcr-cs-ins-abs	*4.70*	**1.30**	**1.30**	*4.50*	*5.20*	2.40	2.30	3.00	*5.05*	*6.90*	*7.75*	*7.60*	3.10	*4.50*	*6.60*	*5.80*
gcr-cs-ins-cb	*5.40*	**1.70**	**1.20**	*5.00*	*4.60*	4.30	2.40	2.10	*4.60*	*6.70*	*7.80*	*7.80*	3.40	*6.20*	*6.60*	2.20

Table 16 Average rankings for 4-objective problems obtained from the Friedman test ($\alpha = 0.05$)

Objectives	SPEA2		NSGA-II		MOEA/D		ϵ-MOEA		GrEA		IBEA		HypE		NSGA-III	
	HV	S	HV	S	HV	S	HV	S	HV	S	HV	S	HV	S	HV	S
gcr-cs-enc-abs	*5.75*	3.05	**1.00**	**2.40**	2.80	4.00	*3.65*	3.35	*5.80*	*7.00*	*7.50*	*7.30*	3.10	5.20	*6.40*	3.70
gcr-cs-enc-cb	*5.50*	**1.50**	**1.20**	*4.10*	*4.40*	*4.00*	*3.90*	3.85	*4.25*	*6.85*	*7.75*	*7.50*	2.60	*4.90*	*6.40*	3.30
gcr-cs-abs-cb	*5.20*	**1.30**	**1.00**	2.90	*4.00*	3.90	2.60	2.20	*5.40*	*7.30*	*7.70*	*7.60*	3.30	*5.90*	*6.80*	*4.90*
gcr-cl-ins-enc	*4.50*	2.20	**1.00**	*6.20*	3.10	4.00	2.40	**1.60**	*6.00*	*6.70*	*7.60*	*7.60*	*4.80*	*4.70*	*6.60*	3.00
gcr-cl-ins-abs	*4.30*	**2.40**	2.00	4.20	*4.80*	2.60	**1.20**	3.00	*5.75*	*6.85*	*7.65*	*7.65*	*3.70*	3.10	*6.60*	*6.20*
gcr-cl-ins-cb	*4.30*	2.40	**1.50**	*5.70*	*5.10*	*4.00*	2.70	**1.30**	*4.60*	*6.90*	*7.80*	*7.80*	3.40	*5.20*	*6.60*	2.70
gcr-cl-enc-abs	*5.60*	3.50	**1.00**	4.90	2.70	3.80	*4.10*	**2.90**	*6.10*	*6.20*	*7.20*	*7.20*	3.10	3.10	*6.20*	4.40
gcr-cl-enc-cb	*5.10*	**2.40**	**1.10**	5.00	*4.60*	3.40	3.20	4.10	*4.30*	*6.90*	*7.80*	*7.60*	3.30	3.80	*6.60*	2.80
gcr-cl-abs-cb	*4.60*	**1.70**	**1.00**	4.00	*5.20*	2.60	3.00	2.60	*5.55*	*7.00*	*7.75*	*7.50*	2.20	*5.00*	*6.70*	*5.60*
gcr-ins-enc-abs	*5.20*	**1.10**	1.70	2.90	3.70	3.70	**1.30**	2.80	*6.40*	*7.20*	*7.60*	*7.60*	3.50	*6.10*	*6.60*	*4.60*
gcr-ins-enc-cb	*5.80*	**1.10**	**1.20**	*3.90*	*4.60*	*4.10*	2.20	2.10	*4.80*	*7.10*	*7.80*	*7.60*	3.20	*6.30*	*6.40*	*3.80*
gcr-ins-abs-cb	*5.60*	**1.10**	**1.30**	3.30	*4.40*	3.50	1.80	2.40	*5.40*	*7.70*	*6.95*	*7.70*	3.00	*6.00*	*6.80*	*5.10*
gcr-enc-abs-cb	*5.50*	**1.75**	**1.10**	2.50	*3.90*	4.20	3.25	3.25	*5.60*	*7.20*	*7.75*	*7.40*	2.40	*5.50*	*6.50*	4.20
cs-cl-ins-enc	*5.20*	**1.80**	**1.70**	*5.30*	4.10	2.90	3.00	3.05	4.30	*6.40*	*7.60*	*7.45*	3.50	*4.60*	*6.60*	*4.50*
cs-cl-ins-abs	3.40	*3.90*	**1.40**	*4.80*	*5.60*	**1.30**	3.50	*5.25*	*4.80*	*5.85*	*7.70*	*7.60*	2.80	*4.50*	*6.80*	2.80
cs-cl-ins-cb	3.20	**2.20**	1.70	4.50	*4.90*	2.60	**1.50**	3.30	*5.60*	*7.00*	*7.60*	*7.60*	*4.60*	3.80	*6.90*	5.00
cs-cl-enc-abs	*5.60*	*5.00*	**1.00**	3.60	2.60	2.50	*4.60*	*5.95*	*5.00*	*5.95*	*7.30*	*6.90*	*3.90*	4.30	*6.00*	**1.80**
cs-cl-enc-cb	*4.50*	*4.65*	**1.50**	4.40	*4.70*	**1.80**	*4.00*	*7.15*	*4.30*	*5.75*	*7.60*	*7.15*	3.20	2.90	*6.20*	2.20
cs-cl-abs-cb	*5.00*	3.40	**1.00**	4.30	*4.40*	2.40	*3.85*	4.30	*5.25*	*6.60*	*7.60*	*7.70*	2.10	*5.30*	*6.80*	**2.00**
cs-ins-enc-abs	*5.60*	**1.30**	1.70	3.10	4.00	*4.30*	**1.60**	1.90	*6.00*	*6.75*	*7.70*	*7.75*	2.80	*6.50*	*6.60*	*4.40*
cs-ins-enc-cb	*5.60*	1.90	**1.50**	*5.10*	*4.20*	3.50	2.30	**1.40**	*4.70*	*6.80*	*7.80*	*7.80*	3.30	*5.60*	*6.60*	3.90
cs-ins-abs-cb	*5.50*	**1.60**	**1.20**	*4.50*	*4.60*	*4.40*	1.90	2.70	*5.45*	*7.00*	*7.75*	*7.80*	2.90	*6.20*	*6.70*	1.80

Table 16 (continued)

Objectives	SPEA2		NSGA-II		MOEA/D		ϵ-MOEA		GrEA		IBEA		HypE		NSGA-III	
	HV	S	HV	S	HV	S	HV	S	HV	S	HV	S	HV	S	HV	S
cs-enc-abs-cb	*5.50*	**1.50**	**1.00**	2.20	3.40	3.70	*4.50*	*5.05*	*5.25*	*6.50*	*7.75*	*7.55*	2.10	*6.10*	*6.50*	3.40
cl-ins-enc-abs	3.80	1.70	1.60	*4.90*	3.40	3.40	**1.40**	**1.50**	*6.20*	*7.45*	*7.75*	*6.85*	*5.20*	*5.20*	*6.65*	*5.00*
cl-ins-enc-cb	*4.40*	2.20	**1.40**	*4.80*	*5.10*	3.30	1.80	**1.20**	*5.00*	*6.85*	*7.65*	*7.75*	*3.90*	*4.20*	*6.75*	*5.70*
cl-ins-abs-cb	*5.50*	1.90	**1.10**	*4.70*	*4.60*	3.20	2.30	**1.30**	*5.30*	*7.00*	*7.65*	*7.80*	*2.70*	*5.60*	2.70	*4.50*
cl-enc-abs-cb	*5.20*	**1.70**	**1.00**	3.00	*3.40*	2.90	*4.00*	*5.85*	*5.10*	*6.75*	*7.65*	*7.30*	*3.20*	*4.60*	*6.45*	3.90
ins-enc-abs-cb	*5.60*	**1.20**	1.80	3.10	*4.40*	*4.20*	**1.40**	2.00	*5.60*	*6.45*	*7.80*	*7.75*	2.80	*6.40*	*6.60*	*4.90*

Table 17 Average rankings for 6-objective problems obtained from the Friedman Test ($\alpha = 0.05$) (cont'd)

Objectives	SPEA2		NSGA-II		MOEA/D		ϵ-MOEA		GrEA		IBEA		HypE		NSGA-III	
	HV	S	HV	S	HV	S	HV	S	HV	S	HV	S	HV	S	HV	S
icd-erp-gcr-cs-cl-ins	*4.50*	**1.70**	**1.00**	*5.00*	*4.20*	3.50	2.10	2.50	*6.15*	*7.30*	*7.75*	*7.60*	*3.70*	*5.40*	*6.60*	3.00
icd-erp-gcr-cs-cl-enc	*5.60*	**1.30**	**1.00**	*4.90*	3.60	*4.70*	3.30	2.20	*5.85*	*7.30*	*7.75*	*7.50*	2.30	*5.10*	*6.60*	3.00
icd-erp-gcr-cs-cl-abs	*5.60*	**1.00**	1.60	3.20	3.60	*5.10*	**1.40**	3.10	*6.00*	*7.40*	*7.80*	*7.60*	3.40	*4.60*	*6.60*	*4.00*
icd-erp-gcr-cs-cl-cb	*5.50*	**1.10**	**1.40**	*4.50*	*4.60*	*4.40*	1.70	2.40	*5.40*	*7.40*	*7.80*	*7.60*	3.00	*5.70*	*6.60*	2.90
icd-erp-gcr-cs-ins-enc	*5.50*	**1.10**	1.80	3.00	3.60	*5.30*	**1.50**	3.10	*5.50*	*7.10*	*7.70*	*7.60*	3.80	*5.00*	*6.60*	*3.80*
icd-erp-gcr-cs-ins-abs	*5.60*	**1.00**	2.10	2.20	3.50	*5.00*	**1.10**	3.30	*5.90*	*7.20*	*7.80*	*7.60*	3.40	*6.10*	*6.60*	3.60
icd-erp-gcr-cs-ins-cb	*5.80*	**1.10**	1.70	3.10	3.70	*5.00*	**1.40**	3.20	*5.25*	*7.10*	*7.75*	*7.50*	3.90	*6.00*	*6.50*	3.00
icd-erp-gcr-cs-enc-abs	*5.60*	**1.00**	2.80	2.30	3.20	*5.10*	**1.00**	*3.80*	*6.10*	*7.40*	*7.70*	*7.60*	3.00	*5.10*	*6.60*	*3.70*
icd-erp-gcr-cs-enc-cb	*5.90*	**1.00**	**1.70**	2.80	3.80	*5.20*	**1.70**	3.20	*5.40*	*7.20*	*7.80*	*7.60*	3.40	*5.30*	*6.30*	*3.70*
icd-erp-gcr-cs-abs-cb	*6.10*	**1.00**	2.10	2.00	*4.10*	*5.20*	**1.30**	3.40	*5.70*	*7.10*	*7.80*	*7.60*	2.80	*5.90*	*6.10*	*3.80*
icd-erp-gcr-cl-ins-enc	*4.20*	**1.70**	**1.30**	*5.10*	*3.90*	*4.40*	1.70	2.80	*6.60*	*7.40*	*7.60*	*7.60*	*4.10*	4.20	*6.60*	2.80
icd-erp-gcr-cl-ins-abs	*4.80*	**1.00**	1.70	*3.30*	3.80	*5.00*	**1.30**	*4.30*	*6.70*	*7.40*	*7.60*	*7.60*	3.50	*3.20*	*6.60*	*4.20*
icd-erp-gcr-cl-ins-cb	*5.50*	**1.10**	1.60	*4.00*	4.10	*4.90*	**1.50**	3.10	*5.80*	*7.40*	*7.80*	*7.60*	3.10	*5.40*	*6.60*	2.50
icd-erp-gcr-cl-enc-abs	*5.40*	**1.00**	2.00	*3.30*	3.40	*4.70*	**1.10**	*4.90*	*6.40*	*7.40*	*7.60*	*7.60*	3.50	*3.30*	*6.60*	*3.80*
icd-erp-gcr-cl-enc-cb	*5.60*	**1.00**	**1.50**	*4.20*	4.00	*4.80*	1.60	3.10	*5.80*	*7.30*	*7.80*	*7.60*	3.10	*5.30*	*6.60*	2.70
icd-erp-gcr-cl-abs-cb	*5.90*	**1.00**	1.70	2.40	*4.20*	*5.20*	**1.30**	*3.50*	*5.75*	*7.40*	*7.75*	*7.60*	3.00	*5.10*	*6.40*	*3.80*
icd-erp-gcr-ins-enc-abs	*5.50*	**1.00**	2.60	2.20	3.00	*4.00*	**1.00**	*4.80*	*6.25*	*7.30*	*7.65*	*7.60*	3.40	5.30	*6.60*	*3.80*
icd-erp-gcr-ins-enc-cb	*5.90*	**1.10**	1.90	2.50	3.80	*5.10*	**1.30**	*4.10*	*5.70*	*7.10*	*7.80*	*7.60*	3.30	*5.00*	*6.30*	3.50
icd-erp-gcr-ins-abs-cb	*5.80*	**1.40**	2.40	2.00	*3.80*	*4.80*	**1.10**	3.60	*5.80*	*7.20*	*7.80*	*7.60*	2.90	*5.70*	*6.40*	3.70
icd-erp-gcr-enc-abs-cb	*6.00*	**1.00**	2.50	2.00	*4.00*	*5.00*	**1.10**	*4.20*	*6.00*	*7.20*	*7.80*	*7.60*	2.40	*5.50*	*6.20*	*3.50*
icd-erp-cs-cl-ins-enc	*5.60*	**1.00**	**1.50**	*3.70*	3.90	*5.70*	1.70	2.20	*5.20*	*7.40*	*7.70*	*7.60*	3.80	*4.70*	*6.60*	*3.70*
icd-erp-cs-cl-ins-abs	*5.60*	**1.00**	2.10	2.30	3.70	*5.60*	**1.10**	2.90	*6.05*	*7.40*	*7.65*	*7.60*	3.20	*4.60*	*6.60*	*4.60*

Table 17 (continued)

Objectives	SPEA2		NSGA-II		MOEA/D		ϵ-MOEA		GrEA		IBEA		HypE		NSGA-III	
	HV	S	HV	S	HV	S	HV	S	HV	S	HV	S	HV	S	HV	S
icd-erp-cs-cl-ins-cb	*5.60*	**1.10**	**1.70**	3.20	*4.60*	*5.10*	1.70	3.00	*4.90*	*7.40*	*7.80*	*7.60*	3.20	*5.20*	*6.50*	3.40
icd-erp-cs-cl-enc-abs	*5.60*	**1.00**	2.00	2.40	3.00	*6.00*	**1.20**	*3.90*	*6.10*	*7.40*	*7.70*	*7.60*	3.80	*3.70*	*6.60*	*4.00*
icd-erp-cs-cl-enc-cb	*5.80*	**1.00**	**1.50**	3.60	*4.40*	*5.40*	1.60	2.70	*5.10*	*7.40*	*7.80*	*7.60*	3.40	*5.30*	*6.40*	3.00
icd-erp-cs-cl-abs-cb	*5.90*	**1.00**	1.80	2.20	*4.20*	*5.40*	**1.40**	3.20	*5.80*	*7.40*	*7.80*	*7.60*	2.80	*5.20*	*6.30*	*4.00*
icd-erp-cs-ins-enc-abs	*5.60*	**1.00**	3.40	2.00	2.90	*5.10*	**1.00**	*3.80*	*6.10*	*7.40*	*7.70*	*7.60*	2.70	*5.70*	*6.60*	3.40
icd-erp-cs-ins-enc-cb	*5.90*	**1.00**	2.00	2.40	*4.30*	*5.00*	**1.30**	3.20	*5.20*	*7.00*	*7.80*	*7.60*	3.20	*6.30*	*6.30*	3.50
icd-erp-cs-ins-abs-cb	*6.10*	**1.00**	2.70	2.10	*4.10*	*4.90*	**1.10**	3.40	*5.90*	*7.10*	*7.80*	*7.60*	2.20	*6.30*	*6.10*	*3.60*
icd-erp-cs-enc-abs-cb	*6.20*	**1.00**	2.60	2.00	*4.00*	*5.20*	**1.20**	*3.90*	*6.00*	*7.10*	*7.80*	*7.60*	2.20	*5.90*	*6.00*	3.30

Table 18 Average rankings for 6-objective problems obtained from the Friedman Test ($\alpha = 0.05$) (cont'd)

Objectives	SPEA2		NSGA-II		MOEA/D		ϵ-MOEA		GrEA		IBEA		HypE		NSGA-III	
	HV	S	HV	S	HV	S	HV	S	HV	S	HV	S	HV	S	HV	S
icd-erp-cl-ins-enc-abs	*5.50*	**1.00**	2.00	2.50	3.30	*5.50*	**1.10**	*4.30*	*6.15*	*7.40*	*7.75*	*7.60*	3.60	3.20	*6.60*	*4.50*
icd-erp-cl-ins-enc-cb	*5.60*	**1.00**	1.60	2.50	*4.30*	*5.40*	**1.40**	2.90	*5.20*	*7.40*	*7.80*	*7.60*	3.50	*5.20*	*6.60*	*4.00*
icd-erp-cl-ins-abs-cb	*5.60*	**1.00**	2.00	2.00	*4.00*	*5.80*	**1.00**	*3.50*	*6.00*	*7.40*	*7.80*	*7.60*	3.00	*4.40*	*6.60*	*4.30*
icd-erp-cl-enc-abs-cb	*5.80*	**1.10**	1.90	1.90	3.60	*5.80*	**1.10**	*4.40*	*6.00*	*7.40*	*7.80*	*7.60*	3.40	*4.00*	*6.40*	*3.80*
icd-erp-ins-enc-abs-cb	*6.00*	**1.00**	2.90	2.00	*3.80*	*4.90*	**1.00**	*4.10*	*6.00*	*7.20*	*7.80*	*7.60*	2.30	*6.10*	*6.20*	3.10
icd-gcr-cs-cl-ins-enc	*5.60*	**1.40**	**1.10**	3.60	*4.10*	*5.10*	2.40	2.30	*4.95*	*7.40*	*7.75*	*7.60*	3.50	*5.00*	*6.60*	3.60
icd-gcr-cs-cl-ins-abs	*5.60*	**1.10**	1.80	2.60	*4.20*	*5.20*	**1.20**	3.30	*5.65*	*7.40*	*7.75*	*7.60*	3.20	*4.70*	*6.60*	*4.10*
icd-gcr-cs-cl-ins-cb	*5.60*	**1.00**	**1.60**	3.50	*4.60*	*5.10*	1.80	2.80	*5.00*	*7.30*	*7.80*	*7.70*	3.00	*5.30*	*6.60*	3.30
icd-gcr-cs-cl-enc-abs	*5.60*	**1.00**	2.10	2.40	3.30	*5.70*	**1.00**	*4.20*	*6.10*	*7.40*	*7.70*	*7.60*	3.60	*3.90*	*6.60*	*3.80*
icd-gcr-cs-cl-enc-cb	*5.60*	**1.00**	**1.20**	*4.20*	*4.40*	*5.00*	2.20	2.30	*4.80*	*7.30*	*7.80*	*7.60*	3.40	*5.00*	*6.60*	*3.60*
icd-gcr-cs-cl-abs-cb	*5.60*	**1.00**	1.70	2.50	*4.30*	*5.40*	**1.60**	3.40	*5.70*	*7.40*	*7.80*	*7.60*	2.70	*4.80*	*6.60*	*3.90*
icd-gcr-cs-ins-enc-abs	*5.60*	**1.00**	3.20	2.00	3.40	*4.90*	**1.00**	*3.60*	*6.10*	*7.30*	*7.70*	*7.50*	2.40	*6.20*	*6.60*	*3.50*
icd-gcr-cs-ins-enc-cb	*5.60*	**1.00**	2.30	2.60	4.40	*4.90*	**2.10**	2.70	*5.10*	*6.90*	*7.80*	*7.60*	2.10	*6.10*	*6.60*	*4.20*
icd-gcr-cs-ins-abs-cb	*5.70*	**1.00**	2.60	2.30	*4.10*	*4.90*	**1.10**	3.30	*5.90*	*7.10*	*7.80*	*7.60*	2.30	*6.30*	*6.50*	3.50
icd-gcr-cs-enc-abs-cb	*5.80*	**1.20**	2.50	1.80	4.00	*4.90*	1.80	3.60	*6.00*	*7.10*	*7.80*	*7.60*	**1.70**	*6.30*	*6.40*	3.50
icd-gcr-cl-ins-enc-abs	*5.60*	**1.10**	2.00	2.20	3.40	*5.20*	**1.10**	*4.90*	*6.15*	*7.40*	*7.65*	*7.60*	3.50	3.20	*6.60*	*4.40*
icd-gcr-cl-ins-enc-cb	*5.60*	**1.00**	**1.50**	3.20	*4.20*	*5.40*	1.60	3.00	*5.00*	*7.40*	*7.80*	*7.60*	3.70	*5.00*	*6.60*	3.40
icd-gcr-cl-ins-abs-cb	*5.60*	**1.00**	1.80	2.20	*4.10*	*5.60*	**1.20**	3.30	*5.80*	*7.40*	*7.80*	*7.60*	3.10	*4.60*	*6.60*	*4.30*
icd-gcr-cl-enc-abs-cb	*5.70*	**1.00**	1.60	2.20	3.70	*5.70*	**1.40**	*4.40*	*6.00*	*7.40*	*7.80*	*7.60*	3.30	*4.00*	*6.50*	*3.70*
icd-gcr-ins-enc-abs-cb	*5.70*	**1.00**	3.10	2.00	3.90	*4.70*	**1.40**	*3.80*	*6.00*	*7.00*	*7.80*	*7.60*	1.60	*6.40*	*6.50*	*3.50*
icd-cs-cl-ins-enc-abs	*5.60*	**1.10**	2.00	2.70	*3.60*	*5.90*	**1.00**	3.20	*6.05*	*7.40*	*7.75*	*7.60*	3.40	*4.20*	*6.60*	*3.90*
icd-cs-cl-ins-enc-cb	*5.60*	1.80	2.20	*4.30*	*4.60*	*3.90*	**1.90**	**1.20**	*4.80*	*7.40*	*7.80*	*7.60*	2.50	*5.70*	*6.60*	*4.10*
icd-cs-cl-ins-abs-cb	*5.60*	**1.20**	2.30	2.60	*4.20*	*5.50*	**1.10**	3.20	*5.55*	*7.30*	*7.75*	*7.70*	2.80	*5.50*	*6.70*	3.00

Table 18 (continued)

Objectives	SPEA2		NSGA-II		MOEA/D		ϵ-MOEA		GrEA		IBEA		HypE		NSGA-III	
	HV	S	HV	S	HV	S	HV	S	HV	S	HV	S	HV	S	HV	S
icd-cs-cl-enc-abs-cb	*5.60*	**1.00**	1.90	2.70	*4.00*	*5.60*	**1.30**	2.60	*5.90*	*7.40*	*7.80*	*7.60*	2.90	*5.00*	*6.60*	*4.10*
icd-cs-ins-enc-abs-cb	*5.60*	**1.00**	3.10	2.00	*4.00*	*4.70*	**1.30**	3.20	*5.90*	*6.90*	*7.80*	*7.40*	1.70	*6.70*	*6.60*	*4.10*
icd-cl-ins-enc-abs-cb	*5.60*	**1.40**	2.00	1.80	*3.90*	*5.90*	**1.00**	3.40	*5.80*	*7.40*	*7.75*	*7.60*	3.30	*4.60*	*6.65*	3.90
erp-gcr-cs-cl-ins-enc	*5.50*	**1.30**	**1.10**	*4.50*	3.20	*5.50*	2.20	2.60	*6.15*	*7.30*	*7.65*	*7.60*	3.60	*4.90*	*6.60*	2.30
erp-gcr-cs-cl-ins-abs	*5.50*	**1.00**	**1.30**	*4.80*	3.70	*3.80*	1.90	*3.60*	*6.00*	*7.30*	*7.80*	*7.60*	3.20	*3.90*	*6.60*	*4.00*
erp-gcr-cs-cl-ins-cb	*5.50*	**1.60**	**1.60**	4.00	4.20	*5.10*	1.70	2.80	*5.80*	*7.35*	*7.80*	*7.65*	2.80	*5.50*	*6.60*	2.00
erp-gcr-cs-cl-enc-abs	*5.30*	**1.00**	**1.00**	*3.30*	2.60	*5.90*	3.60	*3.70*	*6.25*	*7.10*	*7.55*	*7.50*	3.20	*4.20*	*6.50*	*3.30*
erp-gcr-cs-cl-enc-cb	*5.60*	**1.30**	**1.00**	2.40	*3.80*	*5.10*	3.00	3.40	*5.80*	*7.40*	*7.80*	*7.60*	2.50	*5.00*	*6.50*	2.40

Table 19 Average rankings for 6-, 8- and 9-objective problems obtained from the Friedman Test ($\alpha = 0.05$)

Objectives	SPEA2		NSGA-II		MOEA/D		ϵ-MOEA		GrEA		IBEA		HypE		NSGA-III	
	HV	S	HV	S	HV	S	HV	S	HV	S	HV	S	HV	S	HV	S
erp-gcr-cs-cl-abs-cb	*5.50*	**1.30**	**1.30**	2.70	*4.10*	*5.10*	2.80	2.20	*5.50*	*7.40*	*7.80*	*7.60*	2.40	*5.50*	*6.60*	*4.20*
erp-gcr-cs-ins-enc-abs	*5.70*	**1.00**	2.60	2.10	2.80	*5.10*	**1.00**	*3.70*	*6.05*	*7.40*	*7.65*	*7.60*	*3.70*	*5.60*	*6.50*	*3.50*
erp-gcr-cs-ins-enc-cb	*6.10*	**1.10**	1.90	2.80	3.30	*5.10*	**1.50**	3.40	*5.50*	*7.20*	*7.80*	*7.60*	3.80	*5.80*	*6.10*	3.00
erp-gcr-cs-ins-abs-cb	*5.80*	**1.00**	2.00	2.30	3.50	*5.10*	**1.40**	3.20	*5.60*	*7.30*	*7.80*	*7.60*	3.50	*5.80*	*6.40*	*3.70*
erp-gcr-cs-enc-abs-cb	*5.70*	**1.10**	**1.50**	2.20	3.00	*5.20*	2.70	3.20	*5.90*	*7.10*	*7.80*	*7.60*	2.90	*5.90*	*6.50*	3.70
erp-gcr-cl-ins-enc-abs	*5.10*	**1.00**	2.00	*3.30*	3.20	*5.10*	**1.00**	*5.10*	*6.70*	*6.30*	*7.60*	*7.60*	*3.80*	*4.00*	*6.60*	*3.60*
erp-gcr-cl-ins-enc-cb	*5.70*	**1.10**	1.60	3.60	3.80	*5.40*	**1.50**	3.20	*5.80*	*7.40*	*7.80*	*7.60*	3.30	*5.30*	*6.50*	2.40
erp-gcr-cl-ins-abs-cb	*5.70*	**1.10**	**1.60**	2.60	4.10	*5.50*	1.60	3.30	*5.90*	*7.30*	*7.80*	*7.60*	2.80	*5.40*	*6.50*	3.20
erp-gcr-cl-enc-abs-cb	*5.50*	**1.00**	**1.30**	2.30	3.50	*5.90*	2.20	*3.60*	*6.10*	*7.40*	*7.80*	*7.60*	3.00	*4.80*	*6.60*	3.40
erp-gcr-ins-enc-abs-cb	*6.10*	**1.10**	2.30	1.90	3.60	*5.30*	**1.10**	*4.00*	*6.00*	*7.30*	*7.80*	*7.60*	3.00	*5.50*	*6.10*	3.30
erp-cs-cl-ins-enc-abs	*5.60*	**1.00**	2.20	2.30	2.90	*6.00*	**1.00**	*4.40*	*6.15*	*7.40*	*7.65*	*7.60*	*3.90*	*3.90*	*6.60*	3.40
erp-cs-cl-ins-enc-cb	*5.90*	**1.10**	1.60	3.10	*4.10*	*5.50*	**1.40**	3.00	*5.20*	*7.35*	*7.80*	*7.65*	3.70	*5.40*	*6.30*	2.90
erp-cs-cl-ins-abs-cb	*5.70*	**1.10**	1.80	2.40	4.10	*5.50*	**1.70**	3.20	*5.80*	*7.40*	*7.80*	*7.60*	2.60	*5.00*	*6.50*	*3.80*
erp-cs-cl-enc-abs-cb	*5.60*	**1.10**	**1.30**	2.00	3.00	*5.90*	2.50	3.70	*6.00*	*7.40*	*7.80*	*7.60*	3.20	*4.90*	*6.60*	3.40
erp-cs-ins-enc-abs-cb	*6.20*	**1.00**	2.80	2.00	3.30	*5.10*	**1.10**	*3.90*	*6.00*	*7.10*	*7.80*	*7.60*	2.80	*6.20*	*6.00*	3.10
erp-cl-ins-enc-abs-cb	*5.70*	**1.10**	2.10	1.90	3.10	*6.00*	**1.00**	*4.30*	*6.00*	*7.40*	*7.80*	*7.60*	*3.80*	*4.30*	*6.50*	3.40
gcr-cs-cl-ins-enc-abs	*5.60*	**1.00**	2.10	2.30	2.80	*5.80*	**1.10**	*4.30*	*6.15*	*7.40*	*7.65*	*7.60*	*4.00*	*3.80*	*6.60*	*3.80*
gcr-cs-cl-ins-enc-cb	*5.80*	**1.10**	**1.60**	3.20	3.90	*5.70*	1.60	2.70	*5.10*	*7.30*	*7.80*	*7.60*	3.80	*5.30*	*6.40*	3.10
gcr-cs-cl-ins-abs-cb	*5.90*	**1.00**	2.60	2.00	3.60	*4.90*	**1.20**	*3.80*	*6.00*	*7.20*	*7.80*	*7.60*	2.60	*6.20*	*6.30*	3.30
gcr-cs-cl-enc-abs-cb	*5.70*	**1.10**	**1.30**	2.30	3.20	*5.80*	2.40	3.20	*6.00*	*7.40*	*7.80*	*7.60*	3.10	*4.60*	*6.50*	*4.00*
gcr-cs-ins-enc-abs-cb	*5.90*	**1.00**	2.60	2.00	3.60	*4.90*	**1.20**	*3.80*	*6.00*	*7.20*	*7.80*	*7.60*	2.60	*6.20*	*6.30*	3.30
gcr-cl-ins-enc-abs-cb	*5.70*	**1.00**	1.80	2.00	3.30	*6.00*	**1.20**	*4.30*	*6.00*	*7.40*	*7.80*	*7.60*	3.70	*4.30*	*6.50*	3.40

Table 19 (continued)

Objectives	SPEA2		NSGA-II		MOEA/D		ϵ-MOEA		GrEA		IBEA		HypE		NSGA-III	
	HV	S	HV	S	HV	S	HV	S	HV	S	HV	S	HV	S	HV	S
cs-cl-ins-enc-abs-cb	*5.70*	**1.30**	2.10	2.40	3.50	*5.60*	**1.20**	3.10	*5.80*	*7.40*	*7.80*	*7.60*	3.40	*5.30*	*6.50*	3.30
icd-erp-gcr-cs-cl-ins-enc-abs	*5.80*	**1.00**	3.20	2.20	2.80	*5.80*	**1.00**	*4.90*	*5.80*	*7.30*	*7.80*	*7.70*	3.00	*3.90*	*6.60*	3.20
icd-erp-gcr-cs-cs-ins-enc-cb	*6.00*	**1.00**	1.90	2.80	*4.20*	*5.60*	**1.30**	3.50	*5.65*	*7.40*	*7.75*	*7.60*	2.90	*5.20*	*6.30*	2.90
icd-erp-gcr-cs-cl-ins-abs-cb	*6.30*	**1.30**	2.40	1.90	*3.80*	*5.70*	**1.10**	3.80	*5.75*	*7.10*	*7.85*	*7.70*	2.70	*5.20*	*6.10*	3.30
icd-erp-gcr-cs-cl-enc-abs-cb	*6.50*	**1.80**	2.20	**1.80**	*3.90*	*5.70*	**1.10**	3.90	*5.95*	*7.40*	*7.75*	*7.60*	2.80	*4.80*	*5.80*	3.00
icd-erp-gcr-cs-ins-enc-abs-cb	*6.60*	2.00	3.40	**1.60**	3.50	*5.10*	**1.10**	3.80	*5.90*	*7.00*	*7.70*	*7.50*	2.00	*6.10*	*5.80*	2.90
icd-erp-gcr-cl-ins-enc-abs-cb	*6.20*	**1.00**	2.40	2.00	3.60	*5.70*	**1.00**	*4.30*	*5.75*	*7.30*	*7.85*	*7.70*	3.00	*4.80*	*6.20*	3.20
icd-erp-cs-cl-ins-enc-abs-cb	*6.60*	**1.10**	2.80	1.90	3.20	*6.00*	**1.00**	*4.20*	*5.95*	*7.40*	*7.75*	*7.60*	3.00	*4.60*	*5.70*	3.20
icd-gcr-cs-cl-ins-enc-abs-cb	*6.30*	**1.30**	2.60	1.90	3.40	*5.80*	**1.10**	*4.10*	*6.00*	*7.40*	*7.80*	*7.60*	2.90	*4.50*	*5.90*	3.40
erp-gcr-cs-cl-ins-enc-abs-cb	*6.70*	1.80	3.00	**1.60**	2.90	*5.80*	**1.00**	4.00	*5.75*	*7.20*	*7.85*	*7.70*	3.10	*4.80*	*5.70*	3.10
icd-erp-gcr-cs-cl-ins-enc-abs-cb	*6.70*	**1.70**	2.90	1.80	3.30	*5.30*	**1.00**	*4.60*	*5.50*	*6.60*	*7.90*	*7.80*	2.80	*5.00*	*5.90*	3.20

References

Adra S, Fleming P (2011) Diversity management in evolutionary many-objective optimization. IEEE Trans Evol Comput 15(2):183–195

Aleti A, Buhnova B, Grunske L, Koziolek A, Meedeniya I (2013) Software architecture optimization methods: a systematic literature review. IEEE Trans Softw Eng 39(5):658–683

Arcuri A, Briand L (2011) A practical guide for using statistical tests to assess randomized algorithms in software engineering. In: Proceedings of the 33rd international conference on software engineering (ICSE'11), pp 1–10. IEEE

Assunção WKG, Colanzi TE, Vergilio SR, Pozo A (2014) A multi-objective optimization approach for the integration and test order problem. Inf Sci 267:119–139

Bader J, Zitzler E (2011) HypE: an algorithm for fast hypervolume-based many-objective optimization. Evol Comput 19(1):45–76

Bansiya J, Davis CG (2002) A hierarchical model for object-oriented design quality assessment. IEEE Trans Softw Eng 28(1):4–17

Bosch J, Molin P (1999) Software architecture design: evaluation and transformation. In: Proceedings of IEEE conference and workshop on engineering of computer-based systems (ECBS'99), pp 4–10

Boussaïd I, Lepagnot J, Siarry P (2013) A survey on optimization metaheuristics. Inf Sci 237:82–117

Bouwers E, Correia J, van Deursen A, Visser J (2011) Quantifying the analyzability of software architectures. In: Proceedings of the 9th working IEEE/IFIP conference on software architecture (WICSA'11), pp 83–92

Bowman M, Briand LC, Labiche Y (2010) Solving the class responsibility assignment problem in object-oriented analysis with multi-objective genetic algorithms. IEEE Trans Softw Eng 36 (6):817–837

Coello Coello CA, Lamont GB, Van Veldhuizen DA (2007) Evolutionary algorithms for solving multi-objective problems, 2nd edn. Springer

Deb K (2001) Multi-objective optimization using evolutionary algorithms. Wiley, New York

Deb K, Jain H (2014) An evolutionary many-objective optimization algorithm using reference-point-based nondominated sorting approach, Part I: solving problems with box constraints. IEEE Trans Evol Comput 18(4):577–601

Deb K, Mohan M, Mishra S (2003) Towards a quick computation of well-spread pareto-optimal solutions. In: Evolutionary multi-criterion optimization of LNCS, vol 2632. Springer, pp 222–236

Deb K, Pratap A, Agarwal S, Meyarivan T (2002) A fast and elitist multiobjective genetic algorithm: NSGA-II. IEEE Trans Evol Comput 6(2):182–197

del Sagrado J, del Águila IM, Orellana FJ (2015) Multi-objective ant colony optimization for requirements selection. Empir Softw Eng 20(3):577–610

Derrac J, García S, Molina D, Herrera F (2011) A practical tutorial on the use of nonparametric statistical tests as a methodology for comparing evolutionary and swarm intelligence algorithms. Swarm Evol Comput 1(1):3–18

Dobrica L, Niemela E (2002) A survey on software architecture analysis methods. IEEE Trans Softw Eng 28(7):638–653

Ducasse S, Pollet D (2009) Software architecture reconstruction: a process-oriented taxonomy. IEEE Trans Softw Eng 35(4):573–591

Durillo JJ, Zhang Y, Alba E, Harman M, Nebro AJ (2011) A study of the bi-objective next release problem. Empir Softw Eng 16(1):29–60

Garlan D (2000) Software architecture: a roadmap. In: Proceedings of the 22th international conference of software engineering (ICSE'00), pp 91–101

Grunske L (2006) Identifying "Good" architectural design alternatives with multi-objective optimization strategies. In: Proceedings of the 28th international conference on software engineering (ICSE'06), pp 849–852

Gupta P, Verma S, Mehlawat M (2012) Optimization model of COTS selection based on cohesion and coupling for modular software systems under multiple applications environment. In: Computational science and its applications (ICCSA) of LNCS, vol 7335. Springer, pp 87–102

Hadka D, Reed P (2013) Borg: an auto-adaptive many-objective evolutionary computing framework. Evol Comput 21(2):231–259

Harman M, Mansouri SA, Zhang Y (2012) Search-based software engineering: trends, techniques and applications. ACM Comput Surv 45(1):11:1–61

He Z, Yen G, Zhang J (2014) Fuzzy-based Pareto optimality for many-objective evolutionary algorithms. IEEE Trans Evol Comput 18(2):269–285

ISO (2011a) ISO/IEC 25010:2011(E). Software product Quality Requirements and Evaluation (SQuaRE) - System and software quality models. ISO

ISO (2011b) ISO/IEC/IEEE FDIS 42010/D9. Systems and software engineering - Architecture description

Kalboussi S, Bechikh S, Kessentini M, Ben Said L (2013) Preference-based many-objective evolutionary testing generates harder test cases for autonomous agents. In: Proceedings of 5th symposium on search based software engineering (SSBSE'13), pp 245–250

Khare V, Yao X, Deb K (2003) Performance scaling of multi-objective evolutionary algorithms. In: Evolutionary multi-criterion optimization of lecture notes in computer science, vol 2632. Springer, Berlin, pp 376–390

Koziolek A, Ardagna D, Mirandola R (2013) Hybrid multi-attribute QoS optimization in component based software systems. J Syst Softw 86(10):2542–2558

Krogmann K (2010) Reconstruction of software component architectures and behaviour models using static and dynamic analysis. KIT Scientific Publishing

Li R, Etemaadi R, Emmerich MTM, Chaudron MRV (2011) An evolutionary multiobjective optimization approach to component-based software architecture design. In: Proceedings of the IEEE congress on evolutionary computation (CEC'11), pp 432–439

Luna F, González-Álvarez DL, Chicano F, Vega-Rodríguez MA (2014) The software project scheduling problem: A scalability analysis of multi-objective metaheuristics. Appl Soft Comput 15: 136–148

Lutz R (2001) Evolving good hierarchical decompositions of complex systems. J Syst Archit 47(7):613–634

Martin R (1994) OO design quality metrics - An analysis of dependencies. In: Object-Oriented programming systems, languages and applications (OOPSLA), pp 1–8

Mkaouer MW, Kessentini M, Bechikh S, Deb K, Ó Cinnéide M (2014) High dimensional search-based software engineering: finding tradeoffs among 15 objectives for automating software refactoring using NSGA-III. In: Proceedings of the 16th annual genetic and evolutionary computation conference (GECCO'14), pp 1263–1270

Narasimhan VL, Hendradjaya B (2007) Some theoretical considerations for a suite of metrics for the integration of software components. Inf Sci 177(3):844–864

OMG (2010) Unified modeling language 2.4 superstructure specification. OMG. formal/2010-11-14, http://www.omg.org/spec/UML/2.4/

Ouni A, Kessentini M, Sahraoui H, Hamdi MS (2013) The use of development history in software refactoring using a multi-objective evolutionary algorithm. In: Proceedings of the 15th annual genetic and evolutionary computation conference (GECCO'13), pp 1461–1468

Praditwong K, Harman M, Yao X (2011) Software module clustering as a multi-objective search problem. IEEE Trans Softw Eng 37(2):264–282

Praditwong K, Yao X (2007) How well do multi-objective evolutionary algorithms scale to large problems. In: Proceedings of the IEEE congress on evolutionary computation (CEC'07), pp 3959–3966

Purshouse R, Fleming P (2007) On the evolutionary optimization of many conflicting objectives. IEEE Trans Evol Comput 11(6):770–784

Räihä O (2010) A survey on search-based software design. Comput Sci Rev 4(4):203–249

Räihä O, Koskimies K, Makinen E (2011) Generating software architecture spectrum with multi-objective genetic algorithms. In: Proceedings of the 3th world congress on nature and biologically inspired computing (NaBIC'11), pp 29–36

Ramírez A, Romero JR, Ventura S (2014) On the performance of multiple objective evolutionary algorithms for software architecture discovery. In: Proceedings of the 16th annual genetic and evolutionary computation conference (GECCO'14), pp 1287–1294

Ramírez A, Romero JR, Ventura S (2015) An approach for the evolutionary discovery of software architectures. Inf Sci 305:234–255

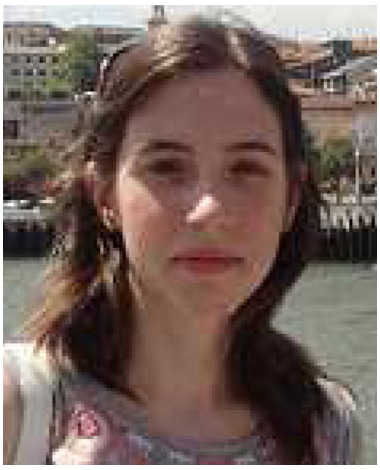
Romano J, Kromrey JD, Coraggio J, Showronek J (2006) Appropriate statistics for ordinal level data: should we really be using t-test and cohen's d for evaluating group differences on the nsse and other surveys? In: Annual meeting of the Florida association of institutional research

Sant'Anna C, Figueiredo E, Garcia A, Lucena CJ (2007) On the modularity of software architectures: a concern-driven measurement framework. In: Software architecture of LNCS, vol 4758. Springer, pp 207–224

Sayyad A, Ammar H (2013) Pareto-optimal search-based software engineering (POSBSE): a literature survey. In: 2nd inter. workshop on realizing artificial intelligence synergies in software engineering (RAISE), pp 21–27

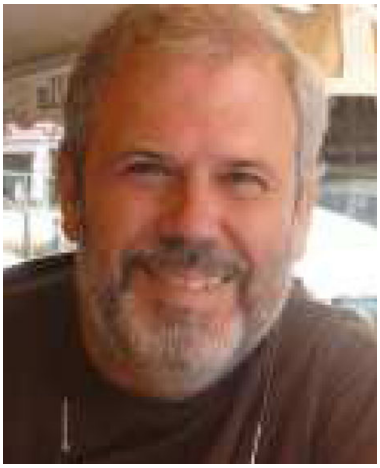
Sayyad AS, Menzies T, Ammar H (2013) On the value of user preferences in search-based software engineering: a case study in software product lines. In: Proceedings of the 35th international conference on software engineering (ICSE'13), pp 492–501

Schutze O, Lara A, Coello Coello CA (2011) On the influence of the number of objectives on the hardness of a multiobjective optimization problem. IEEE Trans Evol Comput 15(4):444–455
Szyperski C (2002) Component software: beyond object-oriented programming, 2nd edn. Addison-Wesley, Boston
Ventura S, Romero C, Zafra A, Delgado JA, Hervás C (2008) JCLEC: a Java framework for evolutionary computation. Soft Comput 12(4):381–392
von Lücken C, Barán B, Brizuela C (2014) A survey on multi-objective evolutionary algorithms for many-objective problems. Comput Optim Appl 58(3):707–756
Wagner T, Beume N, Naujoks B (2007) Pareto-, aggregation-, and indicator-based methods in many-objective optimization. In: Evolutionary multi-criterion optimization of LNCS, vol 4403. Springer, pp 742–756
Wang H, Jiao L, Yao X (2014) An improved two-archive algorithm for many-objective optimization. IEEE Trans Evol Comput. To appear
Yang S, Li M, Liu X, Zheng J (2013) A grid-based evolutionary algorithm for many-objective optimization. IEEE Trans Evol Comput 17(5):721–736
Yao X (2013) Some recent work on multi-objective approaches to search-based software engineering. In: Proceedings of the 5th symposium on search based software engineering (SSBSE), pp 4–15
Zhang Q, Li H (2007) MOEA/D: a multiobjective evolutionary algorithm based on decomposition. IEEE Trans Evol Comput 11(6):712–731
Zhang Y, Harman M, Lim SL (2013) Empirical evaluation of search based requirements interaction management. Inf Softw Technol 55(1):126–152
Zhou A, Qu B-Y, Li H, Zhao S-Z, Suganthan PN, Zhang Q (2011) Multiobjective evolutionary algorithms: a survey of the state of the art. Swarm Evol Comput 1(1):32–49
Zitzler E, Künzli S (2004) Indicator-based selection in multiobjective search. In: Parallel problem solving from nature - PPSN VIII of LNCS, vol 3242. Springer, pp 832–842
Zitzler E, Laumanns M, Bleuler S (2004) A tutorial on evolutionary multiobjective optimization. In: Metaheuristics for multiobjective optimisation of lecture notes in economics and mathematical systems, vol 535. Springer, Berlin, pp 3–37
Zitzler E, Laumanns M, Thiele L (2001) SPEA2: improving the strength Pareto evolutionary algorithm. In: Proceedings of the conference on evolutionary methods for design, optimisation and control with applications to industrial problems, pp 95–100



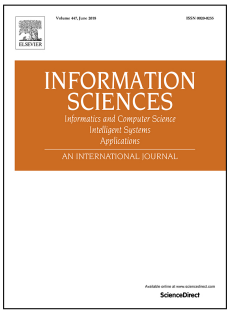
Aurora Ramírez was born in Córdoba, Spain, in 1989. She received a M.Sc. degree in Computer Science from the University of Córdoba, Spain, in 2012. Since 2012, she has been with the Knowledge Discovery and Intelligent Systems Research Laboratory of the University of Córdoba, where she is currently working towards obtaining a Ph.D., and performing research tasks. Her research interests include the application of evolutionary computation on search-based software engineering and data mining.

José Raúl Romero received his Ph.D. in Computer Science from the University of Málaga, Spain, in 2007. He has worked as an IT consultant for important business consulting and technology companies for several years. He is currently an Associate Professor at the Department of Computer Science of the University of Córdoba, Spain. His current research interests include the industrial use of intelligent systems, search-based software engineering, the use of bio-inspired algorithms for data mining, and model-driven software development and its applications. He has published more than 80 papers in journals and scientific conferences. Dr. Romero is a member of the ACM, and the Spanish Technical Normalization Committee AEN/CTN 71/SC7 of AENOR. He can also be reached at http://www.jrromero.net.



Sebastián Ventura is currently an Associate Professor in the Department of Computer Science and Numerical Analysis at the University of Cordoba, where he heads the Knowledge Discovery and Intelligent Systems Research Laboratory. He received his B.Sc. and Ph.D. degrees in Sciences from the University of Cordoba, Spain, in 1989 and 1996, respectively. He has published more than 150 papers in journals and scientific conferences, and he has edited three books and several special issues in international journals. He has also been engaged in 12 research projects (being the coordinator of four of them) supported by the Spanish and Andalusian governments and the European Union. His main research interests are in the fields of machine learning, data mining, computational intelligence and their applications. Dr. Ventura is a senior member of the IEEE Computer, the IEEE Computational Intelligence and the IEEE Systems, Man and Cybernetics Societies, as well as the Association of Computing Machinery (ACM).

6.3. Interactive multi-objective optimisation of software architectures



Title	Interactive multi-objective evolutionary optimization of software architectures
Authors	A. Ramírez, J.R. Romero, S. Ventura
Journal	Information Sciences
Volume	463-464
Pages	92-109
Year	2018
Editorial	Elsevier
DOI	10.1016/j.ins.2018.06.034

IF (JCR 2017)	4.305
Category	Computer Science, Information Systems
Position	12/149 (Q1)

Information Sciences 463–464 (2018) 92–109

Contents lists available at ScienceDirect

Information Sciences

journal homepage: www.elsevier.com/locate/ins

Interactive multi-objective evolutionary optimization of software architectures

Aurora Ramírez, José Raúl Romero*, Sebastián Ventura

Department of Computer Science and Numerical Analysis, University of Córdoba, Córdoba 14071, Spain

ARTICLE INFO

Article history:
Received 5 July 2017
Revised 27 March 2018
Accepted 12 June 2018
Available online 14 June 2018

Keywords:
Search-based software design
Interactive evolutionary computation
Multi-objective optimization
Software architecture discovery

ABSTRACT

While working on a software specification, designers usually need to evaluate different architectural alternatives to be sure that quality criteria are met. Even when these quality aspects could be expressed in terms of multiple software metrics, other qualitative factors cannot be numerically measured, but they are extracted from the engineers know-how and prior experiences. In fact, detecting not only strong but also weak points in the different solutions seems to fit better with the way humans make their decisions. Putting the human in the loop brings new challenges to the search-based software engineering field, especially for those human-centered activities within the early analysis phase. This paper explores how the interactive evolutionary computation can serve as a basis for integrating the humans judgment into the search process. An interactive approach is proposed to discover software architectures, in which both quantitative and qualitative criteria are applied to guide a multi-objective evolutionary algorithm. The obtained feedback is incorporated into the fitness function using architectural preferences allowing the algorithm to discern between promising and poor solutions. Experimentation with real users has revealed that the proposed interaction mechanism can effectively guide the search towards those regions of the search space that are of real interest to the expert.

1. Introduction

Making decisions is an intrinsic aspect of any software design task, since engineers have to choose the best design alternative among all the possibilities on the basis of both functional and non-functional requirements. During the architectural analysis, abstract artifacts need to be precisely identified and specified in order to efficiently guide the development, evolution and deployment of the overall system. Considering such an early stage, architectural decisions become even more challenging due to the lack of knowledge about the system but, at the same time, they are crucial to fulfill the many quality criteria imposed [12].

Artificial intelligence techniques and, more specifically, metaheuristics, can support software engineers in their decision processes by providing them with effective methods to explore a great deal of software designs, each one determined by a different trade-off among the required quality aspects. Such a scenario can be viewed as one of the goals of the search-based software engineering (SBSE) field [14], in which optimization techniques are applied to the resolution of software engineering (SE) tasks conveniently reformulated as search problems. However, solving human-centered activities in a fully

* Corresponding author.
E-mail addresses: aramirez@uco.es (A. Ramírez), jrromero@uco.es (J.R. Romero), sventura@uco.es (S. Ventura).

https://doi.org/10.1016/j.ins.2018.06.034
0020-0255/

automated way seems to be unrealistic, especially for those related to the analysis phase. Certainly, trying to capture the richness of human knowledge only by means of software metrics still represents an unresolved matter to the SE community [32]. Hence, most of the evaluation methods proposed at the architectural level strongly rely on the expert's judgment [10], making extremely difficult to precisely formulate a quantitative fitness function.

Given the relevance of the software architect for the design process, search-based approaches should benefit from his/her knowledge and expertise in order to address the optimization problem in the same way s/he would do it. Interactive optimization [21] constitutes a compelling paradigm here. It allows the human to actively participate in such a way that the expert's opinion can influence both the problem formulation and the search process, allowing the adaptation of the interaction mechanism to the specific requirements of the application domain.

Due to the many various aspects involved with architectural analysis, related optimization problems often require the definition of multiple conflicting objectives. In this sense, the integration of interactive approaches into multi-objective evolutionary algorithms (MOEAs) [7] needs further considerations. Since a MOEA requires the presence of a *decision maker* (DM) in order to choose the final solution among the set of alternatives returned, a logical step would be to allow the DM to dynamically express his/her preferences during the search process [23].

Even when maintaining a multi-objective perspective for architectural optimization is clearly necessary, conceiving an interaction mechanism only founded on expressing opinions about the objective space seems to be insufficient. In addition, comparing several architectural models becomes a hard task due to the information overload. However, engineers would feel more confident when they strictly evaluate qualitative aspects of the automatically generated architectural solutions [31]. This approach permits them to extend the scope of the feedback provided to the algorithm, as well as to adapt the sort of requested opinion as the search elapses. For instance, delivering both positive and negative judgments, which perfectly matches with the human way of acting, can assist the algorithm to discern between interesting and poor solutions with respect to the expert's understanding.

In this context, this paper proposes an interactive evolutionary approach to address the so-called discovery of component-based software architectures [28]. In this problem several software metrics based on maintainability are considered for the evaluation of structural aspects of the components and interfaces that constitute the early software specification. Nevertheless, assessing the adequacy of these highly-abstract software artifacts should also rely on the engineer's feedback. With these factors in mind, the following two research questions (RQ) were stated:

RQ1: How can the qualitative judgment of the engineer be integrated into the evolutionary discovery of software architectures? The proposed interactive approach should consider the multi-objective nature of the optimization problem and define an appropriate evaluation mechanism, in which both qualitative and quantitative evaluation criteria could be put together.

RQ2: Does putting the human in the loop involve a significant improvement compared with not considering him/her along the optimization process? The interactive system should strike a balance between the evolutionary performance, measured by usual quality indicators, and the practical incentive for the software engineer in terms of a reasonable number of high-quality solutions satisfying his/her preferences. Such an analysis requires conducting an empirical study with a substantial number of participants, where aspects like usefulness and intuitiveness should be also evaluated.

A main contribution of this work is the combination of qualitative and quantitative evaluation criteria. On the one hand, an interactive system manages design decisions entered by the engineer to either intensify the search towards specific regions of the search space or, on the contrary, escape from those that do not meet his/her expectations. More specifically, each design decision is mapped into a function, named *architectural preference*, that reinforces the fitness value of solutions satisfying the corresponding qualitative characteristic. On the other hand, the quantitative evaluation in terms of software metrics is kept as a means for achieving promising candidate solutions from a multi-objective perspective. Additionally, it serves to control the inherent uncertainty that arise when dealing with human reasoning, such as fatigue and inconsistency [25]. Reports obtained from real user experiences show that the interactive algorithm here presented is able to adapt the search as new design decisions are made.

The rest of the paper is structured as follows. Section 2 briefly introduces software architecture optimization methods, as well as the concepts and terminology related to the interactive evolutionary computation (IEC) and its application to SBSE. Section 3 describes the optimization problem under study, while the interactive evolutionary approach for discovering software architectures is detailed in Section 4. Next, Section 5 presents the empirical method and experimental framework. Experiments assessing both the evolutionary performance and the applicability of the approach are presented in Section 6. Finally, threats to validity are discussed in Section 7, and Section 8 concludes.

2. Background

This section explains the basis of how search techniques have been previously applied to address architectural design problems, describing some non-interactive optimization approaches. However, interactive optimization systems propose a completely different perspective to face optimization problems, involving the human in the search process. This fact clearly influences their design and implementation, briefly discussed in this section too. Next, some related work focused on the use of interactive approaches in SBSE is introduced.

2.1. Software architecture optimization

During early analysis, the conception of a software architecture satisfying both the functional and non-functional requirements constitutes an important activity. Apart from describing the abstract structure of the system, an architecture exposes the design principles that should guide its subsequent development and evolution [13]. Similarly, architectural models represent essential artifacts when addressing other activities of the software life cycle, such as resource allocation during deployment or reconstruction as part of maintenance and migration [11]. Carrying out these tasks as optimization problems is the idea behind the application of software architecture optimization methods [1].

Optimization methods like metaheuristics can be used to arrange elements of an architectural specification or semi-automatically derive new models. This is done according to predefined quality attributes and other existing constraints. The advantage of applying these methods lies on their high capacity to explore a wide set of design alternatives, only requiring minor adaptation in order to properly manage the problem-specific decision variables. Nevertheless, the abstract and cross-cutting nature of software architectures need to be thoroughly observed to satisfactorily support the decision-making process [12], in which it may influence other factors like human intuition, conflicting goals or the uncertainty inherent to this early stage.

Software architecture optimization has recently emerged as an upward trend in SBSE, providing the necessary support to software engineers when dealing with complex design scenarios. Recent advances reveal that multi-objective evolutionary algorithms can be effectively applied to enhance architectural artifacts at different stages of the design process. For instance, NSGA-II has served to assist engineers in the production of architectural documentation [9]. A hybrid approach considering analytical optimization and a variant of NSGA-II was also presented to cope with the selection and allocation of software components during deployment [16]. Similarly, reconfiguration after deployment was defined as a 5-objective optimization problem to be solved by a specific genetic algorithm [38].

2.2. Interactive optimization

Interactive optimization encompasses all those search methods in which a human explicitly takes part in the search [21]. The need for involving the human within the process can be motivated by many different factors, such as the inability to capture complex features around the problem formulation or the lack of an appropriate quantitative fitness function. This latter issue constitutes a major concern when attempting to solve creative tasks by means of evolutionary computation (EC), so it is not surprising that initial efforts were mostly focused on leaving the responsibility for evaluating candidate solutions to humans [36]. In these cases, showing a subset of the population and then interpolating the fitness for the rest of individuals is a common strategy to reduce the cognitive burden.

When addressing multi-objective problems (MOPs), DMs are expected to establish the desired trade-off among objectives either at the beginning of, during or after the search. To this end, several methods has been proposed, including the negotiation of the importance of each objective and the definition of reference points [23]. The gathered information would be used to redirect the search towards certain regions of the Pareto front (PF) or even to learn from the DM's preferences [5]. Although these mechanisms have been already integrated into some existing MOEAs, other algorithms specifically conceived to deal with MOPs from an interactive perspective can be also found in the literature. A representative approach is iTDEA (*interactive territory defining evolutionary algorithm*) [15], which progressively delimits preferred regions of the PF around the most interesting solutions. More specifically, some solutions are presented to the DM at certain moments of the search process in order to choose the best. According to the feedback obtained, iTDEA updates the size of the territory associated to similar solutions, which determines the permitted distance between the individuals stored in an external archive.

Notice that interactive multi-objective optimization usually restricts the human's decisions to the objective space. However, humans may feel uncertainty about their own opinion when the definition of the objective functions is not easily understandable. Therefore, contributing with opinions on qualitative criteria would fit better in those cases where the interest lies on aspects of the solution to be evaluated from the expert's point of view. This approach is considered in [6], where subjective criteria are considered to define an objective function that is computed together with another function determined by quantitative criteria. In each iteration, the expert rates each solution using a 0–9 scale and can perform additional actions such as altering solutions. Other ways of integrating qualitative information are based on fuzzy modeling of user's preferences and rule-based systems [36].

2.3. Interactive approaches in SBSE

Software engineering seems to represent a natural scenario for interactive optimization, since most of its activities are traditionally carried out by human beings. In fact, real experiences reported by recent works confirm the suitability of these kind of methods [18,34], showing the interest of the SE community in the development of decision support systems under the SBSE paradigm. A recent review of the state-of-the-art [27] also highlights that evaluation mechanisms based on qualitative preferences, whether they have been freely expressed by the expert or selected from a set of options, are preferred over the direct assignment of fitness values.

Software engineers' abilities and know-how are specially important when tackling analysis and design tasks. Therefore, it makes sense that the majority of the interactive approaches proposed in SBSE belong to the so-called search-based software

design subfield [26]. For instance, interactive conceptual object-oriented design has been successfully addressed using both EC [33] and ant colony optimization (ACO) [34]. Here, the authors examine the influence of aesthetic criteria, defined in terms of elegance metrics, when class diagrams are derived from use cases. This work was then extended to allow the designer to freeze parts of the solution, demonstrating the effectiveness of ACO to obtain high-quality solutions after a small number of iterations.

Architecture synthesis [37] and software refactoring [24] are other examples of design problems addressed using interactive approaches. On the one hand, the interaction mechanism employed in [37] allows the architect to freeze classes and design patterns in order to compose the low-level architecture of a software system. Using class diagrams to represent the solutions, they are only evaluated in terms of quantitative software metrics. On the other hand, a multi-objective approach is used in [24] with the aim of improving code quality. Firstly, NSGA-II is responsible for approximating the whole PF. Then, an interactive mechanism assists the engineer in identifying the most interesting refactoring sequences, as they represent the input information required by a local search procedure.

3. Problem fundamentals

This section describes in detail the search problem for the evolutionary discovery of software architectures. Adopted from our previous work [28], where an initial non-interactive evolutionary solution was proposed, the encoding of candidate solutions and the metrics for their quantitative evaluation are also explained.

3.1. The search problem

Understanding the original architecture of a system as it evolves becomes a complex task if the corresponding analysis information is not properly generated and maintained. There are also situations in which the software engineer just needs to specify software artifacts at a higher level of abstraction as an important step prior to the addition of new functionality or the migration of the system. Software components represent abstract units of construction providing well-defined services that can be accessed through their interfaces [35]. Promoting reusability is the ultimate goal of organizing the system structure this way.

In this context, the discovery of component-based software architectures consists in identifying the high-level structure of a software system, in terms of its components and interfaces, from a previous analysis model represented by a UML 2 class diagram [28]. More specifically, the discovery process can be defined according to the following rules:

- A component is derived from a cohesive group of classes, all of them working together in order to implement its behavior.
- Directed relationships among classes belonging to different components serve to identify interfaces, since they represent the provision or need of functionalities. Public methods within classes are used to determine the interface operations. When two or more classes are involved in an interaction between a pair of components, all their public methods would specify a unique interface. Notice that the previous model described in [28] established that each pair of related classes represents a candidate interface. In contrast, the approach here presented has been improved to increase flexibility in such a way that, if the same class is required by other components, its public methods are properly separated following reusability criteria.
- A connector represents the linking between different components in terms of their matching interfaces. As a consequence of how interfaces are now derived, a connector could link a provided interface to several required interfaces.

Following these rules, the search algorithm is able to explore the decision space looking for the optimal allocation of classes and their relationships into components and interfaces. Even so, the abilities and know-how of the software engineer play a key role when envisioning the main functionalities of large software systems and their mutual interactions. Thus, the discovery of software architectures is still a human-centered and iterative task.

3.2. Encoding and initialization

Each candidate solution represents a complete component-based software architecture, whose phenotypic expression corresponds to its representation as a UML 2 component diagram. As for the search, each solution is encoded using a tree structure, whose hierarchical composition perfectly reflects how an artifact, e.g. a component, is comprised of elements of a lower level, e.g. classes and interfaces.

All the encoded solutions should satisfy a number of constraints in order to represent feasible architectural models, as defined next:

- Each class must be only located into one component. Components cannot be empty.
- Components should define at least one interface, either required or provided.
- A pair of components could not provide services to each other, so that they would be mutually dependent.

The population is initialized by arbitrarily distributing classes from the input diagram into a random number of components within the range set by the software engineer. The process controls that classes are not replicated and no empty

Table 1
Software metrics to evaluate component-based architectures.

$ICD = 1/n \cdot \sum_{i=1}^{n} ((\#cl_t - \#cl_i)/\#cl_t) \cdot \left(CI_i^{in}/(CI_i^{in} + CI_i^{out})\right)$
$ERP = \sum_{i=1}^{n} \sum_{j=i+1}^{n} (w_{as} \cdot \#as_{ij} + w_{ag} \cdot \#ag_{ij} + w_{co} \cdot \#co_{ij} + w_{ge} \cdot \#ge_{ij})$
$GCR = \#cgroups/n$

component is returned. However, constraints regarding inadmissible interactions among components are not checked. These unfeasible individuals will be penalized in terms of their fitness value, making them progressively disappear.

3.3. Software metrics for quantitative evaluation

Dealing with several quality attributes like modularity or reusability in the same design is a usual situation for the architect. Hence, the discovery of software architectures can be addressed from a multi-objective perspective, where quality metrics measuring such attributes need to be simultaneously optimized. Table 1 shows the three metrics, i.e. objectives, used in this work:

- *Intra-modular Coupling Density* (ICD) looks for a trade-off between coupling and cohesion. For each component i it calculates the ratio between internal (CI_i^{in}) and external (CI_i^{out}) relations, also considering the number of inner classes ($\#cl_i$). $\#cl_t$ stands for the total number of classes in the full model, and n is the number of components. This metric has to be maximized, and varies in the range [0,1].
- *External Relations Penalty* (ERP) counts the number of relationships between classes of different components, i and j, that cannot be declared with a well-defined interface. This situation mostly occurs when the navigability of associations (*as*), aggregations (*ag*) or compositions (*co*) is not explicitly defined in the source analysis model. Generalization relationships (*ge*) between classes located in different components, e.g. representing a data abstraction, may also turn into external dependencies. The software architect can specify how detrimental a dependency caused by each kind of relationship is for the resulting architecture in terms of their respective weights (w_x). ERP should be minimized, 0 being the optimum.
- *Groups/Components Ratio* (GCR) defines the ratio between the number of groups of interconnected classes (*#cgroups*) and the number of components in the entire architecture (n). A well-defined component is expected to contain a unique group of classes, so the optimal value is 1. This metric should be minimized.

4. Interactive model for the evolutionary discovery of software architectures

As stated in *RQ1*, considering the feedback from the software engineer into the discovery of software architectures would permit adding new valuable qualitative information to the search process. With this aim, this section provides an overview of the proposed IEC model. Then, a more detailed description of its essential elements is presented, including the solution evaluation method based on both quantitative and qualitative criteria, the mechanisms enabling the management and transformation of solutions, and how human interaction is conducted.

4.1. Overview of the approach

Fig. 1 shows the proposed evolutionary model (henceforth named iMOEA), which is composed of two main elements: the algorithm conducting the automatic search of architectural models, and the interaction module that coordinates the communication between the algorithm and the software engineer. More specifically, the multi-objective evolutionary algorithm here proposed is based on a steady-state scheme, and makes use of a sophisticated diversity preservation technique similar to hyperboxes. Both aspects are relevant according to our previous findings [30], since they provided a more appropriate convergence and a control mechanism to reduce the archive size, respectively. In addition, the algorithm defines a specific evaluation method that combines quantitative (software metrics) and qualitative (architectural preferences) criteria. The resulting fitness function is then used as one criterion to compare solutions along the search process.

Firstly, the algorithm initializes the population according to the procedure detailed in Section 3.2. Notice that, at that time, solutions can be evaluated by only meeting quantitative criteria, i.e. the software metrics serving as objectives (see Section 3.3). An initial archive is also created from the set of non-dominated solutions. Then, the evolutionary search follows the usual execution flow: parent selection, genetic operators, replacement and archive update. The search continues until a stopping condition is met.

At certain moments of the evolution, the algorithm momentarily stops the search to obtain feedback from the software engineer. During the interaction, the algorithm selects a subset of the population to be evaluated in terms of qualitative criteria. Architectural preferences are then defined in such a way that the algorithm will be able to numerically determine to what extent each candidate solution satisfies these criteria. The engineer can also perform additional *actions*, such as freezing some specific elements of the architectural model under evaluation or even stopping the search. Notice that his/her choice might influence the course of the evolution in steps like the generation of offspring. After the first interaction, the evaluation phase begins to consider both quantitative and qualitative criteria, incorporating the expert's perspective in the optimization process.

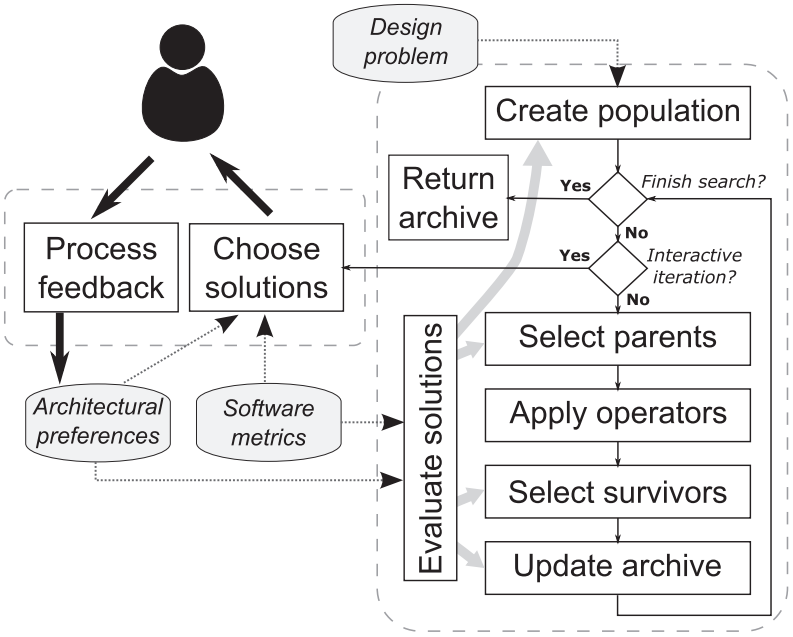


Fig. 1. Proposed interactive evolutionary model.

4.2. Fitness function: putting together human decisions and software metrics

The lack of consistency and the cognitive burden, which are inherent in any interactive system, become even more critical when dealing with software architectures due to the presence of highly abstract artifacts. All this, combined with the huge amount of solutions that an algorithm can generate in one single execution, implies that relying only on the engineer's judgment to assess the quality of the solutions would be impractical. Thus, an effective evaluation mechanism requires striking the right balance between objective and subjective criteria. Software design metrics have proven to be effective in identifying the overall functional blocks of the architecture [28]. As the design progresses and a more fine-tuned design is required, the participation of the expert becomes more relevant. Therefore, a reward/penalization approach [27] is adopted to capture the engineer's expectations. Given that both qualitative and quantitative assessments should be combined and computed, the feedback provided by the expert should be then mapped into numerical preference functions. With this aim, Eq. (1) defines the fitness function of a solution s as a weighted sum of two terms, f_{obj} and f_{sub}. Weights w_{obj} and w_{sub} are considered in order to let the engineer control the relative importance of the objective and subjective evaluation, respectively. This function varies in the range [0,1] and should be minimized. Next, each component of the fitness function is explained in detail.

$$fitness(s) = w_{obj} \cdot f_{obj}(s) + w_{sub} \cdot f_{sub}(s) \tag{1}$$

4.2.1. Objective evaluation: software metrics

The *objective component*, f_{obj}, requires the definition of a set of software metrics, each one representing a conflicting objective. Without limiting the generality, f_{obj} considers that all of them should be minimized and vary in the range [0,1]. Consequently ERP and GCR have been scaled accordingly. Their theoretical upper limit will depend on the number of relationships and the number of classes contained by the source analysis model, respectively. In addition, ICD values need to be inverted.

Given that obtaining a weighted sum of the objective values would target the search towards a unique point in the PF, a different kind of aggregation function is required here in order to transform the metric information into a single value. With this aim, f_{obj} uses the *maximin* function [4]. For a given set of objectives $k \in [1, K]$ this function returns a value in the range $[-1,1]$, reporting on both the dominance and the diversity of a solution with respect to a reference set Z, e.g. the whole population. On the one hand, the sign of the result serves to distinguish between non-dominated (<0), weakly-dominated ($=0$) and dominated solutions (>0). On the other hand, the specific value gives an idea of the proximity between non-dominated solutions, values close to -1 being preferred. For a dominated solution, the result represents its distance to the PF, values close to 0 meaning proximity. Both properties serve to precisely qualify a candidate solution from a multi-objective perspective and, at the same time, the obtained values can be easily interpreted. Eq. (2) shows the formulation of the *maximin* function, properly adapted to return a value in the range [0,1], as required by Eq. (1). f_k^s represents the value of the objective k for the solution under evaluation (s), whereas f_k^z represents the same value but for a solution, z, belonging to the reference set (Z). Notice that the *maximin* function inspects all the search directions in order to find which one allows

the solution s to be far from being dominated.

$$f_{obj}(s) = (1 + max_{z \neq s}(min_k(f_k^s - f_k^z)))/2 \ \forall z \in Z \tag{2}$$

4.2.2. Subjective evaluation: architectural preferences

To quantify the *subjective component*, f_{sub}, the algorithm makes use of the set of design decisions compiled after each interaction. Taking a candidate solution as a reference, the engineer might highlight a qualitative aspect that he/she considers relevant to appear in a final solution, or focus his/her attention on features to be avoided. Notice that qualitative criteria will mostly be focused on phenotypic aspects of a solution, e.g. whether a software component is meaningful. Once the association between the design decision and the architectural preference has been established, the algorithm is responsible for promoting the solutions that satisfy positive preferences and, at the same time, penalizing those presenting undesirable characteristics according to the negative preferences. To do this, $pref_p$ in Eq. (3) measures to what extent an arbitrary solution s satisfies the architectural preference p. The returning value, i.e. the *degree of achievement*, lies in the range [0,1] and should be maximized, regardless of whether the engineer's opinion is positive or negative. In addition, each preference has an associated weight, w_p, which represents the engineer's confidence in his/her decision. This can be expressed using the Likert-type scale, though the corresponding weight should be scaled in order to ensure that the result remains in the range [0,1]. Here, weights are relative to a unique interaction, so the specific value is computed according to the confidence levels associated to all the evaluations made in the same interaction.

$$f_{sub}(s) = 1 - 1/P \cdot \sum_{p=1}^{P} (w_p \cdot pref_p(s)) \tag{3}$$

At different interactions, the expert will express his/her opinion on either the specific composition of the solutions under evaluation or the values of the returned software metrics. The following list compiles the preference alternatives available for the engineer:

1. *No preference*. The expert skips providing any opinion.
2. *Best component*. For a given solution, the engineer selects the best component, c^+, according to its structure. A preference function $pref_{bc}$ will determine to which extent there are other individuals having a similar component (see Eq. (4)). For each component c of a solution, the function $cl()$ extracts its classes, so that the resulting set is compared with the set of classes contained in c^+. With this aim, the Jaccard index J is calculated (see Eq. (5)). Given a pair of sets, A and B, this similarity measure calculates the ratio between the number of common elements and the number of different elements. Finally, the maximum similarity value among the n components comprising the architectural solution is returned as the degree of achievement of this preference.

$$pref_{bc} = max\{J(cl(c), cl(c^+))\} \ \forall c \in [1, n] \tag{4}$$

$$J(A, B) = |A \cap B|/|A \cup B| \tag{5}$$

3. *Worst component*. In contrast to the previous preference, this preference allows the engineer to express a negative opinion on an observed component c^-. Here, the corresponding preference function $pref_{wc}$ penalizes those solutions having a component similar to c^- (see Eq. (6)).

$$pref_{wc} = max\{1 - J(cl(c), cl(c^-))\} \ \forall c \in [1, n] \tag{6}$$

4. *Best provided interface*. The expert may identify an interface p^+ of interest for the service interaction specification, even when the component providing it is not properly formed yet. Similar to $pref_{bc}$, the preference function defined by Eq. (7) computes the Jaccard index, in this case being used to compare sets of interface operations. The function $op(i)$ serves to extract the operations from an interface i, while $in(c)$ compiles the interfaces provided by a component c.

$$pref_{bi} = max\{J(op(in(c)), op(p^+))\} \ \forall c \in [1, n] \tag{7}$$

5. *Worst provided interface*. Similar to $pref_{wc}$, it focuses on interfaces instead of components. Eq. (8) shows the expression that calculates the preference function $pref_{wi}$, p^- being the rejected interface.

$$pref_{wi} = max\{1 - J(op(in(c)), op(p^-))\} \ \forall c \in [1, n] \tag{8}$$

6. *Number of components*. The engineer may be interested in leading the search for solutions with a preferred number of components n^+. In this case, $pref_{nc}$ calculates the difference between this number and the current number of components of the given solution, n. In Eq. (9), $n_{\min}$ and $n_{\max}$ are the limits initially set by the engineer to the size of the architecture. An evolutionary consequence, which remains transparent to the expert, is the obvious reduction of the search space.

$$pref_{nc} = \begin{cases} (n - n_{min})/(n^+ - n_{min}) & \text{if } n < n^+ \\ 1 - ((n - n^+)/(n_{max} - n)) & \text{if } n \geq n^+ \end{cases} \tag{9}$$

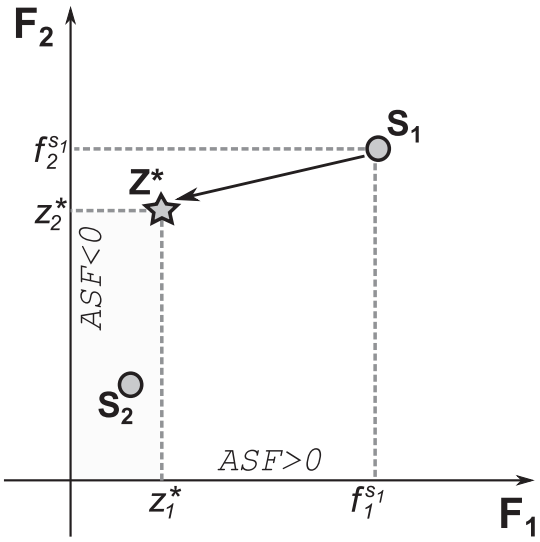


Fig. 2. An illustrative example of ASF values.

7. *Metric in a range*. This preference helps the engineer to determine the expected values—or range of values—of a given metric m (see Table 1 in Section 3.3). Having set a maximum ($m_{\max}$) and a minimum ($m_{\min}$) value for m, the preference $pref_{mr}$ penalizes any solution s with a metric value m^s outside this interval. On the contrary, values close to the midrange (m_{mid}) are rewarded, as shown in Eq. (10).

$$m_{mid} = (m_{max} - m_{min})/2 \tag{10}$$
$$pref_{mr} = \begin{cases} 0 & \text{if } m^s < m_{\min} \\ 1 - (m^s - m_{mid})/m_{mid} & \text{if } m^s \in [m_{\min}, m_{\max}] \\ 0 & \text{if } m^s > m_{\max} \end{cases}$$

8. *Aspiration levels*. This preference allows the engineer to set the target values for all the metrics. From the evolutionary perspective, aspiration levels [23] are appropriate to guide the search towards solutions whose objective values are close to the DM expectations. Achievement scalarizing functions (ASFs) [22] are usually applied to determine to what extent a solution s satisfies the aspiration levels represented in the form of a reference point z^*. The ASF here selected, shown in Eq. (11), computes a weighted distance in each search direction k so that the overall preference $pref_{al}$ promotes those solutions with a small ASF value. Notice that solutions having better objective values, f_k^s, than z^* obtain the maximum degree of achievement for this preference k. Assuming equal weights, w_k, Fig. 2 illustrates both cases. The ASF value for s_1 would be determined by the distance in axis F_1. ASF is lower than 0 for s_2 as it has better objective values than z^*. Both the reference point and the weights are provided by the software engineer.

$$ASF = max\{w_k \cdot (f_k^s - z_k^*)\} \tag{11}$$
$$pref_{al} = \begin{cases} 1 & \text{if } ASF \leq 0 \\ 1 - ASF & \text{if } ASF > 0 \end{cases}$$

4.3. Selection, mutation and replacement strategies

Focusing on the selection mechanism, two parents are selected using binary tournament, one from the population and another from the archive. Since the competition is based on their fitness values, both quantitative and qualitative criteria influence the selection process.

Then, the algorithm applies the mutation operator in order to produce two offspring. The mutation operator simulates five different architectural transformations: adding a component (a); removing a component (r); merging two components (m); splitting a component (s); and moving a class (c) [28]. For each individual, the mutator executes a probabilistic roulette with these operations ensuring that any architectural output will be comprised of an allowed number of components, according to the aforementioned thresholds. After selecting an operation and applying it, any mutant not satisfying all the constraints is discarded, and the original solution mutated once again. This process is performed for a maximum of 10 attempts. If the mutator still fails to find a feasible individual, the initial solution is returned. In addition, the operator should be aware of the fact that individuals can contain frozen parts that should not be modified. On the other hand, crossover is not considered because it would hardly generate feasible solutions [28].

Finally, the replacement strategy causes a competition among offspring and current individuals, promoting the survival of those solutions having better fitness values. Solutions discarded by the engineer are progressively removed from the

population in order to keep the population size constant. Additionally, if three or more solutions are marked to be removed, two of them would be eliminated in the current generation, while the others will be conveniently penalized in order to ensure their removal in future generations.

4.4. Archive update mechanism

Algorithm 1 describes the procedure to update the archive, which is partially based on the definition of territories pro-

Algorithm 1 Update archive.

```
Require: population, archive
1:  archive' ← archive
2:  for all ind ∈ population do
3:    if ind ∉ archive' then
4:      accept ←false
5:      dominated ← solutionsDominatedBy(ind)
6:      r ← preferredRegion(ind)
7:      t ← territorySize(r)
8:      s ← closerSolution(archive', ind)
9:      d ← distance(s, ind)
10:     if ind was selected by the user then
11:       accept ←true
12:       if d < t then
13:         reduceTerritorySize(r, t − d)
14:       end if
15:     else
16:       if ind is non-dominated then
17:         if d > t then
18:           accept ←true
19:         else
20:           n ← numberOverlappingTerritories(ind)
21:           if n == 1 AND f_sub(ind) > f_sub(s) then
22:             accept ←true
23:             archive' ← (archive' ∩ ¬s)
24:           end if
25:         end if
26:       end if
27:     end if
28:   end if
29:   if accept then
30:     archive' ← (archive' ∩ ¬dominated ∪ ind)
31:   end if
32: end for
33: return archive'
```

posed by iTDEA [15]. However, there are some aspects of the original procedure that have been conveniently adapted in order to apply the proposed fitness function when determining which solutions should be kept in the archive after each generation. Firstly, the method checks that the individual was not added beforehand (line 3). Secondly, it extracts the set of dominated solutions (line 5). Then, the method proceeds like iTDEA (lines 6–9). More specifically, a preferred region for the given individual (r) is determined according to a set of weights associated to each objective function [15]. After finding the archive member s closer to the individual (line 8), the rectilinear distance between them is calculated (line 9). At this point, the decision about whether *ind* should replace s is not only based on that distance, but also on how much each one satisfies the engineer's preferences. It is a key difference with respect to iTDEA, which would simply discard *ind* if it lies on the territory associated to s. The complete acceptance criteria are defined as follows:

- A solution of interest to the engineer is always accepted, regardless of whether it is dominated or not (line 11). In addition, they cannot be removed in subsequent generations. The territory size of the associated region is conveniently reduced when the solution lies on the territory of another solution (lines 12–14).
- Replacing one non-dominated solution with another belonging to the same region will be permitted if it implies improving f_{sub} (lines 19–26). It should be noted that if the solution overlaps with the territory of two or more archive members (line 20), the action will not be performed in order to avoid discarding an excessive number of solutions.

- Dominated solutions are only removed when the individual whose acceptance is being checked is finally added (lines 31–33), as suggested in [15].

After each interaction, the algorithm reduces the territory size for the region allocating the best solution in terms of f_{sub}, allowing a higher density of solutions around.

4.5. Interaction mechanism

There are two relevant factors that may affect users' fatigue and their loss of interest: the frequency of interaction and the mechanism of selection of solutions. To prevent fatigue, the engineer is able to set the desired number of interaction steps. Then, how these interactions are distributed along the search is automatically determined according to Koksalan and Karahan [15]. Given a maximum number of generations, g, $g/3$ generations are executed before the first interaction in order to approximate the PF, while $g/6$ iterations are performed by the algorithm after the last interaction to ensure that human decisions are properly propagated. Between these two points, the user will be able to take action at regular intervals.

A clustering approach is applied in order to select the most representative solutions of the overall population. Notice that it is also important to provide software engineers with information regarding the improvements resulting from their decisions. Therefore, if m solutions are required to be displayed to the expert, a *kMeans++* algorithm [3] selects $m-1$ solutions, looking for diversity with respect to their objective values. The remaining solution is the one with the highest f_{sub} value.

Engineers are also allowed to perform additional actions. Firstly, the most promising solutions can be directly saved to the archive, not only implying that they will be returned at the end of the search process, but also that they could be selected as parents more frequently. Secondly, solutions not satisfying the engineer's expectations could be identified for removal in the replacement phase. Finally, parts of an individual genotype could be frozen as a way to facilitate their appearance in other solutions.

5. Experimental framework

The IEC model has been coded in Java using JCLEC-MOEA [29]. In addition, some supporting libraries have been used to process data,[1] extract analysis information from XMI files[2] and implement the clustering procedure.[3] These algorithm executions aimed at assessing the performance of the evolutionary search were run on an Ubuntu 16.4 computer with 8 cores Intel Core i7 2.67-GHz and 7.79-GB RAM.

The rest of this section explains the empirical methodology conducted to properly respond to research questions *RQ1* and *RQ2*. The parameter set-up and problem instances are detailed next.

5.1. Methodology

The performance of a new MOEA is usually assessed in terms of quality indicators. Nevertheless, interactive approaches also imply putting the human in the loop in order to perform their evaluation. Both ways are complementary, though the latter requires conducting some particular experimentation and analysis.

Firstly, as posed in *RQ2*, the performance of the multi-objective evolutionary approach has to be proved before the human getting involved. With this aim, a parameter study is required to analyze the behavior of the algorithm regarding the returned number of solutions and the expected trade-off between their quality and diversity. These properties will be evaluated using two quality indicators, hypervolume (*HV*) and spacing (*S*) [7]. In addition, the evolutionary performance is compared against the well-known NSGA-II algorithm [8]. In both cases, algorithms will be executed 30 times with different random seeds over all the available problem instances.

The significance of the outcomes [2] is assessed by applying non-parametric statistical tests. More specifically, the Wilcoxon Signed-Rank test will be executed to perform pairwise comparisons, while the Aligned Friedman test will be used when the experiment involves more than two algorithms. An effect size measurement is also considered to assess the performance gain, the Cliff's Delta test being selected to this end. For all experiments, the null hypothesis, H_0, establishes that the corresponding algorithms perform equally well at a specific level of significance, 1-α.

Once the non-interactive algorithm (hereinafter referred to as bMOEA, *base MOEA*) is properly analyzed and tuned, the interactive approach (iMOEA) will be empirically assessed through the participation of 9 people. All the participants are either students or professionals in the field of computer science, with previous background in software development for a period of 2–17 years. More specifically, the experiment is conducted by 1 undergraduate student, 2 master students, 4 software engineers (postgraduates) and 2 academics (PhDs), who face a specific real-world problem instance named Datapro4j (see further details in Section 5.2). This software system has been selected as case study because its analysis model could be understandable by the engineer for the time required to conclude the experiment. Additionally, about 33% of the participants already had some previous development experience with Datapro4j in one form or another.

[1] http://www.uco.es/grupos/kdis/datapro4j.
[2] http://www.sdmetrics.com/OpenCore.html.
[3] http://commons.apache.org/proper/commons-math/.

Table 2
Required parameters and their values.

Parameter	*Value*
Population size	150
Maximum number of evaluations	24,000
Minimum number of components	2
Maximum number of components	6
ERP metric weights (w_{as}, w_{ag}, w_{co}, w_{ge})	1, 2, 3, 5
Mutation weights (w_a, w_r, w_m, w_s, w_c)	0.2, 0.1, 0.1, 0.3, 0.3
Fitness weights (w_{obj}, w_{sub})	0.5, 0.5
Initial territory size (τ_0)	0.01, 0.05, 0.1
Final territory size (τ_H)	0.005
Decreasing factor (λ)	0.5
Number of interactions	3
Number of solutions per interaction	3

Interactive sessions have been planned as follows. At the beginning, participants are instructed in the purpose and content of the experiment, as well as in the use of the interactive tool, including the architectural preferences and actions. Given that users are able to visualize the entire architectural model, special considerations are taken to reduce the inherent cognitive burden. On the one hand, internal information about classes is not shown as it remains unaltered from one solution to another, and a printed copy of the input class diagram is also provided during the session. On the other hand, the number of components is restricted according to the configuration parameters, while the mechanism proposed to derive interfaces ensures that the number of interfaces per component is not excessive.

Each participant executes a single run with the best configuration obtained after the parameter study and a different random seed. The interaction scheme consists of 3 stops for the user to evaluate 3 different solutions each time. This way all the participants evaluate the same number of solutions and every interactive execution will provide intermediate results under the same conditions.

Exhaustive execution logs are generated to properly study the behavior of participants and receive relevant feedback from their experience. During the experiment, participants are requested to write down the reasons of their decisions. Similarly, at the very end of the session, they should fill in a form with additional questions about their experience and impressions, as well as free comments or suggestions.

The two planned experiments, i.e. the parameter study and the empirical investigation, serve to respond *RQ2*. On the one hand, the analysis in terms of quality indicators proves the competitiveness of the algorithm from an evolutionary point of view. On the other hand, the usefulness and intuitiveness of the approach can be derived from the information gathered from the interactive session. Furthermore, the influence of the participants' opinion with respect to the sort of the solutions found and the level of metric optimization is assessed by comparing the outcomes of bMOEA and iMOEA. Notice that, in this problem, any configuration is a solution, with the only exception of unfeasible individuals. Therefore, the fitness function is intended to simulate the software engineer's behavior in terms of his/her design preferences, which could not be formulated beforehand, as they are expected to be indicated as the search progresses. Consequently, the solutions returned do not necessarily have to be Pareto optimal, but a reflection close to the expectations of the expert.

5.2. Algorithm set-up

Table 2 shows the list of parameters and their respective values. Overall parameters like the population size and the stopping criteria, as well as those being specific to the problem under study, have been set according to our previous studies [28,30]. Same weights were applied to the objective and subjective evaluations. Furthermore, the influence of the territory size (τ) on the update mechanism of the archive deserves special attention, since it might affect the number of resulting solutions. In this case, three different initial values of τ are tested during the non-interactive experiment, while recommendations from the authors of iTDEA are followed regarding the mechanism to update it after each interaction [15]. As mentioned in Section 5.1, the interaction scheme ensures that all users interact 3 times with the algorithm, sequentially showing 3 different solutions each time.

Table 3 shows the quantitative characteristics of each problem instance. They provide a wide spectrum of complexity regarding the number of classes and relationships, which are classified into associations (*as*), aggregations (*ag*), compositions (*co*), generalizations (*ge*) and dependencies (*de*). The last column indicates the number of candidate interfaces, i.e. the number of relationships whose navigability has been explicitly specified. With the exception of Aqualush,[4] a benchmark used for educational purposes, all the problem instances represent working software systems.[5]

[4] http://www.ifi.uzh.ch/en/rerg/research/aqualush.html.
[5] http://www.java-source.net/.

Table 3
Problem instances and their characteristics.

Problem	*Classes*	*Relationships*					*Interfaces*
		as	*ag*	*co*	*ge*	*de*	
Aqualush	58	69	0	0	20	6	74
Datapro4j	59	3	3	2	49	4	12
Java2HTML	53	20	15	0	15	66	170
JSapar	46	7	21	9	19	33	80
Marvin	32	5	22	5	8	11	28
NekoHTML	47	6	15	18	17	17	46

Table 4
Number of solutions of the final archive.

	$\tau = 0.01$	$\tau = 0.05$	$\tau = 0.1$
Aqualush	23.77 ± 6.28	14.80 ± 2.52	9.23 ± 1.87
Datapro4j	12.57 ± 3.62	8.37 ± 2.06	5.73 ± 1.12
Java2HTML	140.37 ± 3.03	49.75 ± 5.17	17.48 ± 1.92
JSapar	21.67 ± 4.99	12.20 ± 2.94	8.00 ± 1.41
Marvin	13.27 ± 4.02	8.27 ± 2.02	5.93 ± 1.29
NekoHTML	20.07 ± 4.56	11.60 ± 3.20	7.97 ± 1.33

Table 5
Results for spacing (S).

	$\tau = 0.01$	$\tau = 0.05$	$\tau = 0.1$
Aqualush	0.039 ± 0.023	0.045 ± 0.028	0.048 ± 0.020
Datapro4j	0.038 ± 0.034	0.044 ± 0.045	0.050 ± 0.027
Java2HTML	0.021 ± 0.002	0.022 ± 0.004	0.030 ± 0.009
JSapar	0.041 ± 0.021	0.064 ± 0.017	0.063 ± 0.031
Marvin	0.031 ± 0.009	0.029 ± 0.015	0.035 ± 0.019
NekoHTML	0.033 ± 0.011	0.032 ± 0.017	0.037 ± 0.019
Ranking	14.500	9.500	4.500

6. Analysis of results

6.1. Evolutionary performance

The number of solutions returned to the expert is a key aspect when dealing with real-world decision scenarios and interactive approaches. In the proposed bMOEA, the archive size can be controlled by the parameter τ. Even though the final number of solutions might depend on the way the engineer interacts with the algorithm, providing some guidance to the algorithm in this regard could help.

As explained in Section 5.2, the algorithm has been run considering three possible values of τ: 0.01, 0.05 and 0.1. Notice that there is no interaction here, so τ remains constant along the search. Consequently, it only imposes a restriction with respect to the minimum number of solutions to be returned. Table 4 shows the average archive size for the three aforementioned configurations, including the standard deviation. As might be expected, increasing the value of τ allows reducing the archive size. Furthermore, the specific problem instance might be another determinant factor.

Given that a limited number of solutions could compromise the expected trade-off between convergence and diversity, a further analysis of the quality of the solutions is carried out in terms of *HV* and *S*. Firstly, the spacing values for the three configurations are studied in order to confirm that the use of territories allows our bMOEA to effectively preserve diversity while maintaining an affordable number of alternatives to be returned to the participant.

Table 5 shows the obtained results for each problem instance, where higher values are better. As can be seen, setting too low values of τ might demote the expected diversity, mainly because of the increase of the number of solutions returned by the algorithm. Rankings reported by the Aligned Friedman test are also included in Table 5. To confirm the existence of statistical differences using this test, the statistic z–distributed according to a chi-square distribution with 2 degrees of freedom–should be compared to a critical value of that distribution. Given that the obtained statistic, $z = 4.1477$, is smaller than the critical value (5.9915) for $\alpha = 0.05$, H_0 cannot be rejected, that is, bMOEA behaves similarly in terms of spacing for the three proposed configurations.

In contrast, as can be observed in Table 6, when comparing the three configurations in terms of *HV*, better values are now obtained for $\tau = 0.01$, though the Aligned Friedman test reveals that there are not statistical differences ($z = 4.0730$) at a confidence level (CL) of 95%. Consequently, $\tau = 0.05$ is chosen as the most appropriate value for our algorithm, since it

Table 6
Results for hypervolume (*HV*).

	$\tau = 0.01$	$\tau = 0.05$	$\tau = 0.1$
Aqualush	0.637 ± 0.012	0.620 ± 0.015	0.599 ± 0.017
Datapro4j	0.658 ± 0.017	0.639 ± 0.017	0.614 ± 0.022
Java2HTML	0.282 ± 0.026	0.278 ± 0.027	0.261 ± 0.035
JSapar	0.555 ± 0.015	0.552 ± 0.018	0.530 ± 0.018
Marvin	0.616 ± 0.008	0.614 ± 0.007	0.604 ± 0.012
NekoHTML	0.602 ± 0.012	0.586 ± 0.015	0.567 ± 0.015
Ranking	3.667	9.333	15.500

Table 7
Results obtained by NSGA-II.

	Num. of solutions	*Spacing*	*Hypervolume*
Aqualush	147.73 ± 11.84	0.018 ± 0.008	0.635 ± 0.015
Datapro4j	148.67 ± 4.25	0.013 ± 0.013	0.645 ± 0.010
Java2HTML	148.87 ± 2.96	0.009 ± 0.008	0.404 ± 0.048
JSapar	150.00 ± 0.00	0.018 ± 0.007	0.547 ± 0.014
Marvin	140.60 ± 23.22	0.014 ± 0.005	0.618 ± 0.009
NekoHTML	150.00 ± 0.00	0.016 ± 0.007	0.596 ± 0.013

Table 8
Architectural preferences applied during the experiment.

Architectural preference	*% Selected*	*Usefulness*	*Intuitiveness*
No preference	22.22%	6.44	7.33
Best component	29.63%	7.44	7.44
Worst component	23.46%	7.22	7.33
Best provided interface	2.47%	5.29	6.38
Worst provided interface	0.00%	4.71	6.38
Number of components	17.28%	7.50	7.33
Metric in range	2.47%	4.17	5.44
Aspiration levels	2.47%	5.80	5.22

achieves the best trade-off between both indicators and the difference with the corresponding best possible configuration is not statistically significant.

In addition to the influence of τ, it would be also interesting to find out to what extent the proposed algorithm provides a similar performance that those MOEAs aimed at returning an approximation of the whole PF. In fact, NSGA-II has also been considered here due to its ability to effectively guide the search towards non-dominated solutions and the lack of an explicit mechanism to limit the number of final solutions. Table 7 shows the results for *S* and *HV*, as well as the number of non-dominated solutions found. Although NSGA-II provides values of *HV* slightly higher than our algorithm (see Table 6 for $\tau = 0.05$), it is worth noticing the difference with respect to the spacing indicator. Besides, the high number of returned solutions could complicate the decision-making process.

Pairwise comparison is performed here in order to precisely compare their performance. Regarding *HV*, the Wilcoxon test reveals that NSGA-II performs better than our algorithm with a CL of 90% ($p - value = 0.0938$), even though this difference is classified as *negligible* by the Cliff's Delta test. On the contrary, our proposal is statistically better than NSGA-II in terms of *S* with a CL of 95%, $p - value = 0.0313$, the difference between both algorithms being classified as *large*. In this sense, our algorithm is a competitive alternative against NSGA-II, since it is able to find not only high quality but also representative solutions, even when the archive size has to be limited.

6.2. Use of interactive mechanisms

A key aspect of the proposed interactive approach is that the engineer is able to evaluate the solutions provided by the algorithm in qualitative terms, e.g. by indicating both positive and negative preferences that might influence the subsequent search process. It is interesting to observe how these architectural preferences are selected, and how useful and intuitive participants consider their application. These two factors are scored by users on a scale between 1 (*poor*) and 8 (*excellent*). Table 8 shows the frequency of use of preferences, and the average rating for usefulness and intuitiveness.

Within the preferences group, participants generally focus their attention on the internal structure of the components. In fact, indicating the preferred number of components that should comprise the architectural specification is also a frequent choice. In contrast, preferences related to interfaces have been rarely selected. It is likely that users consider a priority to find a proper structure at first, and they omit any further detail on the component interaction, even when it could be a factor to refine the search by filling components with the most appropriate interacting constituents, i.e. classes and relationships.

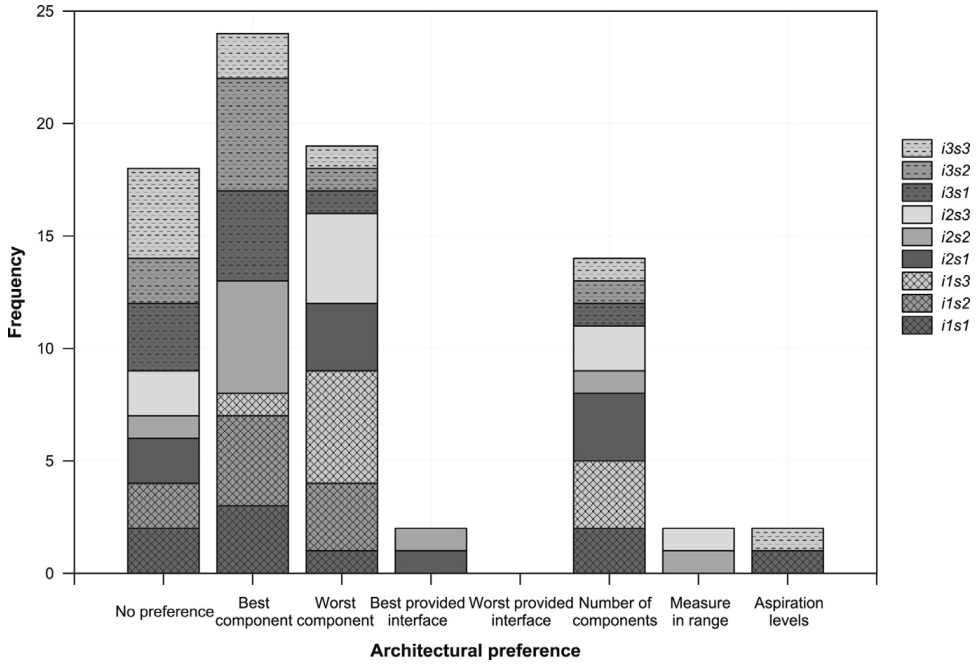


Fig. 3. Frequency and moment of selection of each architectural preference.

Table 9
Other actions taken during the interactive session.

Optional action	*% Selected*	*Usefulness*	*Intuitiveness*
Add to archive	34.78%	6.11	6.89
Remove from population	30.43%	5.89	6.89
Freeze components	34.78%	7.44	7.44
Stop search	0.00%	5.14	7.44

Participants also avoid setting specific values to software metrics, possibly because they consider this could not lead to such a straightforward and tangible result. Finally, it is also a common practice not to indicate any architectural preference. Some participants pointed out that they just wanted to observe how the algorithm could evolve by itself, whilst others found it a convenient way to deal with uncertainty about making a precise judgment. In any case, the applicability of architectural preferences seems to be related to their intuitiveness and usefulness, according to users' scores.

In order to examine the behavior of the participants, it is also interesting to study whether their design decisions are somehow related to the interaction moment. Fig. 3 shows the total number of occurrences of the different preferences at each specific interaction point, where *i* stands for the number of interaction break, and *s* for the solution position in the group of three. Notice that, apart from the omission of choice, the three most intuitive and useful preferences—according to the user rating—were applied throughout the entire search process. Nevertheless, some additional patterns can be still observed. For instance, during the initial interactions, users tend to express negative opinions (e.g. *worst component*) or to indicate some overall restriction (e.g. *number of components*) in order to reach an expected solution. However, as the evolution progresses, positive opinions become more frequent because better solutions are returned.

Apart from indicating a preference, participants have the opportunity to take additional actions, such as adding the solution to the archive, definitely removing it, or even freezing a specific part of an architecture. They could also stop the search process to get the current archive. It is noticeable how these actions are not frequently selected by the user. Actually, 66% of the participants applied at least one of these actions but only once or twice. Table 9 summarizes how many times each action was taken, and how users rated their usefulness and intuitiveness. Participants never stopped the search, as they did not even find it so useful, probably because of the limits in the number of iterations and time constraints. When applied, adding solutions and freezing components served the participants to reinforce their preferences, which also allowed them to perceive the effect of these actions in subsequent interactions.

6.3. Impact of subjective evaluation

Without the user interaction, the proposed algorithm would be guided by objective measures like other non-interactive approaches. The participation of the user might apparently distort the solutions returned from a merely evolutionary perspective. Nevertheless, notice that the real power of the iMOEA lies in its ability to reinforce certain solutions that engineers might find close to their expectations. This is an important aspect to be considered in domains like search-based software

Table 10
Evolution of software metrics.

	bMOEA			iMOEA		
	ICD	ERP	GCR	ICD	ERP	GCR
Initial population	0.619 ± 0.009	0.716 ± 0.009	0.689 ± 0.010	0.548 ± 0.006	0.712 ± 0.006	0.685 ± 0.006
Final population	0.640 ± 0.035	0.030 ± 0.013	0.027 ± 0.012	0.366 ± 0.066	0.168 ± 0.084	0.165 ± 0.085
Initial archive	0.476 ± 0.021	0.518 ± 0.053	0.491 ± 0.053	0.481 ± 0.036	0.520 ± 0.074	0.483 ± 0.070
Final archive	0.419 ± 0.029	0.133 ± 0.035	0.121 ± 0.033	0.414 ± 0.041	0.147 ± 0.046	0.139 ± 0.042

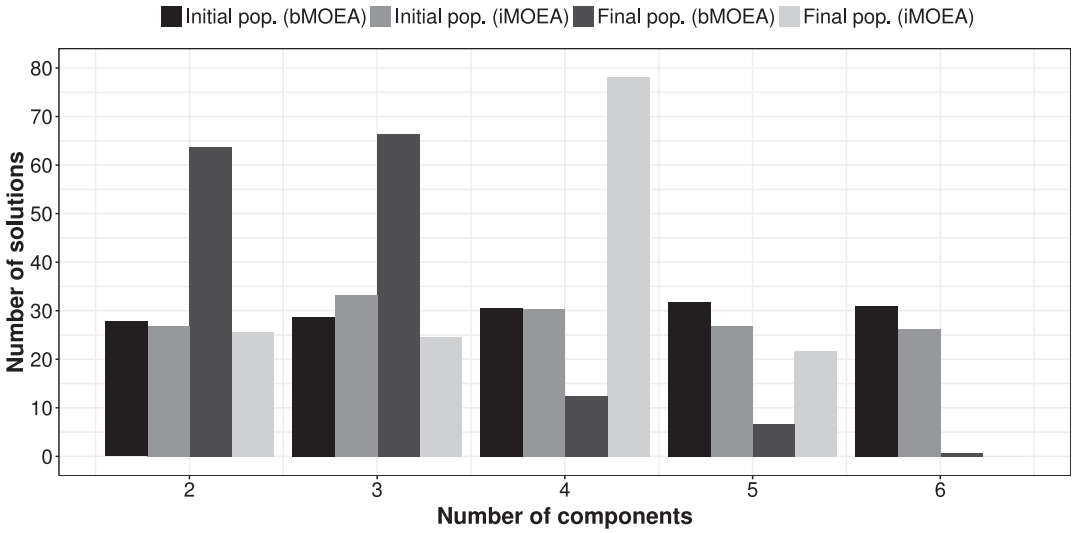


Fig. 4. Variation in the size of the solutions during the evolutionary process.

design, where a number of objective measures is not enough to evaluate the know-how, previous experiences and overall expectations of the user. Next, the actual influence of the subjective decisions made along the interactive process in the quality of the solutions is discussed.

According to the results from Section 6.1, bMOEA and iMOEA behave similarly from an evolutionary perspective. For instance, for iMOEA, the average value of *HV* is equal to 0.6368, which is quite close to the result obtained by bMOEA, $HV = 0.6391$ (see Table 6 for the Datapro4j instance). Even two participants reached higher values for *HV* than bMOEA. In terms of *S*, for iMOEA the 9 values lie on the range [0.0143, 0.0859], the average value ($S = 0.0433$) being a little less than that obtained by bMOEA ($S = 0.0439$). Here, the observed difference in iMOEA is mainly due to the activation of the archive update mechanism, which would be reacting to completely different user actions. In fact, the average number of solutions stored in this set increased from 8.37 to 11.56, returning archives with up to 19 solutions for iMOEA.

Bringing the human in the loop influences the reached trade-off among software metrics. According to the conducted experimentation, most of the design decisions made by the participants were implicitly directed towards the increase of the ICD metric, which otherwise would tend to be demoted by the evolutionary algorithm in favor of ERP and GCR. More specifically, Table 10 shows the average value for each metric in both the initial and the final population. These values are also obtained for the solutions belonging to the archive. It should be noted that all values belong to the range [0,1], 0 being the optimum. For bMOEA, ERP and GCR are greatly improved in the regular population, whereas ICD remains quite constant or even increases. On the contrary, iMOEA is able to reduce ICD significantly without causing a dramatic increase of the other metrics with respect to the values reached by bMOEA. Focusing on the archive, these differences are not so evident. Notice that the archive only stores non-dominated solutions well distributed over the PF, the obtained average values representing better trade-offs among the three metrics. In this sense, it is likely that interesting solutions from the engineer's point of view, e.g. those with low ICD values, represent dominated solutions. This explains why ICD values are higher in the archive than in the regular population. In this case, these solutions cannot appear in the archive unless the user explicitly indicate that they should be added.

With iMOEA, software engineers may also recommend their preferred structure for the architectural solutions. As discussed in Section 6.2, most of the participants selected the architectural preference *number of components* at some point in the interactive session. More precisely, many of them agreed that 4 components was an appropriate value for this problem instance. As can be seen in Fig. 4, this choice drastically affected the evolution, and iMOEA rapidly discarded solutions of other sizes. In fact, notice that, even when both bMOEA and iMOEA start with a similar distribution of solutions regarding their number of components, bMOEA finally leads the search towards architectural solutions having 2 or 3 components. It is caused by the optimization of the objective criteria, since ERP and GCR can be reduced by creating large components.

6.4. Comparison between solutions

A complementary view of the outcomes of the interactive session can be made focusing on the type of solutions found, and their similarity with the manually produced architecture. In addition, it also serves to analyze to what extent participants' decisions might differ from those made by a search process only guided by design metrics. As a reference, the architectural specification of the case study, *Datapro4j*, was originally comprised of 4 components: *Datasets, Columns, Algorithms (a.k.a. Strategies)* and *Datatypes*. Please notice that this comparison is made against the original source, human-designed specification, which should not strictly comply with the design requirements implied by the design metrics, as they could have consider others as well.

As a result of the interactive session, 104 solutions were returned as part of the 9 final archives. From this set, 6 solutions (5.77%) contain a component that is equal to one of those specified by the original designers. These solutions were obtained from 5 different participants (55.56% of the executions), but only one of them explicitly stored the solution in the archive. If approximations to the source architecture are taken into account,[6] the percentage of solutions would increase to 76,92%. In this scenario, all the participants were able to find at least two alternative solutions with these characteristics. In addition, 3 out of 4 original components were approximated at least once, *Algorithms* being the most frequently identified component (in 52 of the 104 solutions). It is worth mentioning that 7 solutions archived by the participants contained that component, which was also frozen in 4 of them. This suggests that the algorithm can achieve 'human-looking' solutions with the assistance of the user, requiring just a few manual modifications to reproduce the original architecture.

After conducting a similar analysis on the non-interactive algorithm, it can be observed that an exact reproduction of at least one original component is found in 18 of the 251 solutions (7.17%). However, they all are generated by only 8 of the 30 executions (26.67%). When approximate components are considered, percentages increase up to 73.31% and 100%, respectively. In this case, the algorithm provides good approximations of 3 components, two of them being the same that those identified by iMOEA. The components appearing more often are those comprised of less classes and presenting less interactions, i.e. *Dataypes* and *Algorithms*, as they can be more easily isolated. Therefore, the interaction with the user could also help the algorithm to find a good separation of the most coupled functionalities.

It is worth mentioning that even when the algorithm is able to find similar solutions to those specified manually, the interactive approach takes advantage of human design abilities to identify core functionalities and thus is able to produce meaningful components. In addition, mechanisms like freezing parts of the solutions and their external storage may reinforce the search in order to rapidly propagate human design decisions. In contrast, the influence of the stochastic nature of the evolutionary process can be mitigated.

6.5. Human experience

In the context of software design, analyzing a number of full architectural models requires a major effort for the expert. Therefore, steps should be taken in order to alleviate the burden of successive evaluations, such as restricting the number of interactions with the algorithm, or reducing the spent time. With this aim, participants were asked to respond to a survey—scores between 1 (*completely disagree*) and 8 (*completely agree*)—related to their fatigue at the end of their interaction. Their replies indicate that, even though they partially disagree with the idea that the interactive session took too long (3.44), they mostly recognized that they were paying more attention at the beginning of the process (5.13). This is clearly reflected by the average time spent per evaluation (see Fig. 5). Notice that users take more time to evaluate solutions shown during the initial interaction, probably because of a greater interest and less knowledge on the problem and the process itself. However, as the search process advances and they acquire more knowledge about candidate solutions, users tend to reduce their degree of interest in exploring new alternatives from the returned solutions, while improving their ability to process the information displayed by the interactive tool. This effect is also noticeable for the first solution shown in every interaction.

As for their feedback, users pointed out that they found promising the idea of having tools to support design tasks (6.89), since they consider that it helps reducing the effort that the design process implies (6.50). The overall perception was that the algorithm could provide interesting solutions (6.13), and it even helped them to discover new design alternatives that they had not thought about (6.38). Finally, participants also made diverse suggestions:

- To extend the number of architectural preferences and actions available, as well as to allow applying more than one preference to the same solution.
- To allow the user to directly manipulate the provided solution, e.g. by splitting components or moving elements.
- To enable participants to undo previous actions and reconsider their decisions, as well as to allow exploring the candidate solutions at once, instead of sequentially.

7. Threats to validity

The experimental and empirical nature of this study places certain limitations, which are discussed next in terms of validity threats.

[6] Here, a component having all the classes of the original component with a margin of error of ± 2 classes has been considered as a valid approximation.

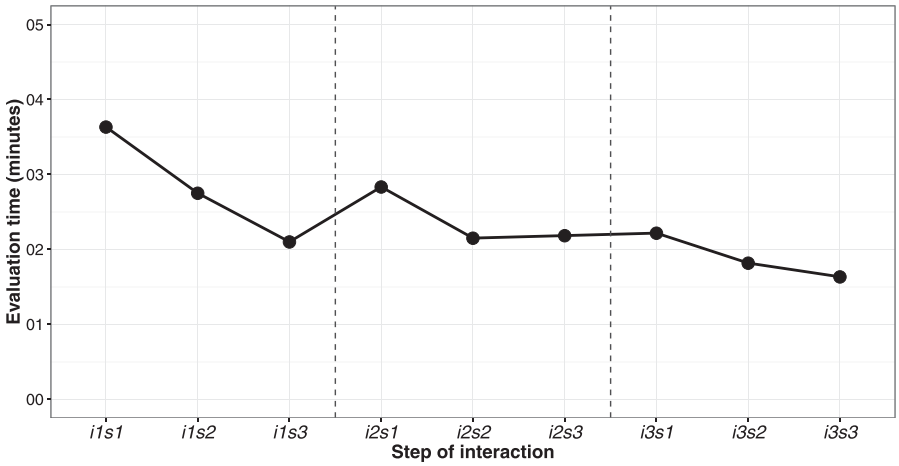


Fig. 5. Average evaluation time during interactions.

Internal validity refers to those aspects of the experimentation that cannot ensure the causality between the hypothesis and the obtained results. In this regard, comparisons between algorithms are based on 30 independent runs in order to deal with the intrinsic randomness of evolutionary algorithms. Appropriate statistical tests have been applied to draw conclusions about the performance of the algorithms in terms of two commonly used quality indicators. As for the experiment putting the human in the loop, the relative small sample size would represent the main threat to internal validity. Nevertheless, 9 or even smaller sizes are common and properly accepted for interactive studies in SBSE [18,34], since the motivation behind this kind of experimentation is mostly focused on the analysis of the human experience, the usefulness of the approach, and the contribution of human-made decisions to the evolutionary process.

The design of the interactive experiment poses additional threats to construct validity. Focusing on the selection of participants, none of them had previous background on the use of interactive algorithms, though some of them were familiar with evolutionary computation in domains different from SBSE. On the contrary, a few participants had some prior knowledge about the system under study, which was intentionally selected in order to reflect the diversity of the practical scenario.

The threat caused by the user fatigue was controlled by applying a fixed interaction scheme, where each participant performs only one execution within an interactive session never longer than 1 h. It also suitable to mitigate the learning effect. Besides, the visualization and evaluation of complete architectural models might produce a heavy cognitive load to the user. The proposed evaluation method tries to overcome this situation by focusing on the qualitative assessment of parts of the solution, and it has been conceived to be open to other complementary mechanisms.

External validity is related to the generalization of the experimental results. Although participants are mostly working in an academic context, some of them have previous experience in the industry or are currently working in an industrial setting. Nevertheless, given that they all are software engineers, the experiment has served to confirm the benefit of the interactive approach as a supporting mechanism to support the SE professional in the understanding of the underlying architecture of a real software system.

8. Concluding remarks

This paper presents an interactive multi-objective evolutionary algorithm aimed at supporting software engineers during the early analysis process. The combination of multi-objective optimization techniques with the so-called architectural preferences guides the search towards the joint optimization of both objective and subjective criteria. Both types of evaluation depend on the specific characteristics of the architecture optimization problem to be addressed, even so the adaptation of the proposed algorithm in order to solve other design tasks would only require the redefinition of specific quality criteria, e.g. the software metrics and architectural preferences. Under the assumption that engineers might detect more easily those model elements that they dislike when analyzing complex architectural specifications, they have been also provided with the possibility to indicate negative opinions on candidate solutions.

As for its validation, the proposed approach has been compared against a well-known multi-objective algorithm like NSGA-II. To study its suitability for bringing the human in the loop, the algorithm has also been validated with a representative number of users with different expertise levels, who have participated in interactive sessions. Results show that the interactive approach is able to manage the expected trade-off between specific requirements of a real decision scenario: good enough solutions, variety of alternatives and a restricted number of solutions. Furthermore, the use of architectural preferences as a mechanism for the subjective, qualitative evaluation helps to overcome the limitations related to the use of numerical ratings, as usually proposed by other IEC approaches.

To conclude, such a human in the loop approach would definitely allow software engineers to actively participate in the generation and evaluation of different design alternatives, providing the search algorithm with accurate information con-

cerning their real expectations and, consequently, leading to more satisfactory results. In the future, we intend to consider the suggestions made by the participants to improve the interactive experience, and analyze whether their abilities to recognize promising parts of a solution might help to improve the search performance. In addition, the proposed evaluation mechanism could be also applied to other multi-objective decision scenarios like the engineering field [17,19], where the opinion and knowledge of experts may suppose a significant difference to reach their expectations [20].

Acknowledgments

This work was supported by the Spanish Ministry of Economy and Competitiveness [projects TIN2014-55252-P, TIN2017-83445-P]; the Spanish Ministry of Education under the FPU program [grant FPU13/01466]; and FEDER funds.

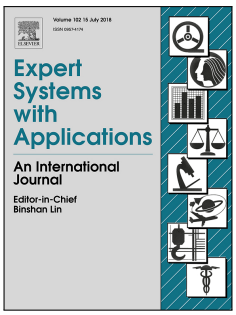
References

[1] A. Aleti, B. Buhnova, L. Grunske, A. Koziolek, I. Meedeniya, Software architecture optimization methods: a systematic literature review, IEEE Trans. Softw. Eng. 39 (5) (2013) 658–683.
[2] A. Arcuri, L. Briand, A hitchhiker's guide to statistical tests for assessing randomized algorithms in software engineering, Softw. Test. Verif. Rel. 24 (3) (2014) 219–250.
[3] D. Arthur, S. Vassilvitskii, k-means++: the advantages of careful seeding, Proc. 18th Ann. ACM-SIAM Symp. Discrete Algorithms (2007) 1027–1035.
[4] R. Balling, S. Wilson, The maximin fitness function for multi-objective evolutionary computation: application to city planning, Proc. Genetic Evol. Comput. Conf. (2001) 1079–1084.
[5] J. Branke, S. Greco, R. Slowinski, P. Zielniewicz, Learning value functions in interactive evolutionary multiobjective optimization, IEEE Trans. Evol. Comput. 19 (1) (2015) 88–102.
[6] A.M. Brintrup, J. Ramsden, H. Takagi, A. Tiwari, Ergonomic chair design by fusing qualitative and quantitative criteria using interactive genetic algorithms, IEEE Trans. Evol. Comput. 12 (3) (2008) 343–354.
[7] C.A.C. Coello, G.B. Lamont, D.A.V. Veldhuizen, Evolutionary Algorithms for Solving Multi-Objective Problems, second ed., Springer, 2007.
[8] K. Deb, A. Pratap, S. Agarwal, T. Meyarivan, A fast and elitist multiobjective genetic algorithm: NSGA-II, IEEE Trans. Evol. Comput. 6 (2) (2002) 182–197.
[9] J.A. Díaz-Pace, C. Villavicencio, S. Schiaffino, M. Nicoletti, H. Vázquez, Producing just enough documentation: an optimization approach applied to the software architecture domain, J. Data Semant. 5 (1) (2016) 37–53.
[10] L. Dobrica, E. Niemela, A survey on software architecture analysis methods, IEEE Trans. Softw. Eng. 28 (7) (2002) 638–653.
[11] S. Ducasse, D. Pollet, Software architecture reconstruction: a process-oriented taxonomy, IEEE Trans. Softw. Eng. 35 (4) (2009) 573–591.
[12] D. Falessi, G. Cantone, R. Kazman, P. Kruchten, Decision-making techniques for software architecture design, ACM Comput. Surv. 43 (4) (2011) 1–28.
[13] D. Garlan, Software architecture: a roadmap, Proc. Conf. Future Software Eng. (2000) 91–101.
[14] M. Harman, S.A. Mansouri, Y. Zhang, Search based software engineering: trends, techniques and applications, ACM Comp. Surv. 45 (1–11) (2012) 1–61.
[15] M. Koksalan, I. Karahan, An interactive territory defining evolutionary algorithm: iTDEA, IEEE Trans. Evol. Comput. 14 (5) (2010) 702–722.
[16] A. Koziolek, D. Ardagna, R. Mirandola, Hybrid multi-attribute qos optimization in component based software systems, J. Syst. Softw. 86 (10) (2013) 2542–2558.
[17] S. Kundu, S. Das, A.V. Vasilakos, S. Biswas, A modified differential evolution-based combined routing and sleep scheduling scheme for lifetime maximization of wireless sensor networks, Soft Comput. 19 (3) (2015) 637–659.
[18] B. Marculescu, R. Feldt, R. Torkar, S. Poulding, An initial industrial evaluation of interactive search-based testing for embedded software, Appl. Soft Comput. 29 (2015) 26–39.
[19] R. Marler, J. Arora, Survey of multi-objective optimization methods for engineering, Struct. Multidiscipl. Optim. 26 (6) (2004) 369–395.
[20] L. Martínez, J. Liu, D. Ruan, J.B. Yang, Dealing with heterogeneous information in engineering evaluation processes, Inf. Sci. 177 (7) (2007) 1533–1542.
[21] D. Meignan, S. Knust, J.M. Frayret, G. Pesant, N. Gaud, A review and taxonomy of interactive optimization methods in operations research, ACM Trans. Interact. Intell. Syst. 5 (3–17) (2015) 1–43.
[22] K. Miettinen, M.M. Mäkelä, On scalarizing functions in multiobjective optimization., OR Spectrum 24 (2) (2002) 193–213.
[23] K. Miettinen, F. Ruiz, A.P. Wierzbicki, Multiobjective optimization: Interactive and evolutionary approaches, chapter Introduction to Multiobjective Optimization: Interactive Approaches, pages 27–57, Springer, Berlin, Heidelberg, 2008.
[24] M.W. Mkaouer, M. Kessentini, S. Bechikh, K. Deb, M.O. Cinnéide, Recommendation system for software refactoring using innovization and interactive dynamic optimization, in: Proc. 29th ACM/IEEE Int. Conf. Automated Software Engineering, pages 331–336, ACM, 2014.
[25] I.C. Parmee, Poor-definition, uncertainty, and human factors - satisfying multiple objectives in real-world decision-making environments, Evol. Multi-Criterion Optim. (2001) 52–66.
[26] O. Räihä, A survey on search-based software design, Comput. Sci. Rev. 4 (4) (2010) 203–249.
[27] A. Ramírez, J.R. Romero, C. Simons, A systematic review of interaction in search-based software engineering, IEEE Trans. Software Eng. (2018).
[28] A. Ramírez, J.R. Romero, S. Ventura, An approach for the evolutionary discovery of software architectures, Inf. Sci. 305 (2015) 234–255.
[29] A. Ramírez, J.R. Romero, S. Ventura, An extensible JCLEC-based solution for the implementation of multi-objective evolutionary algorithms, in: Proc. Companion Publication 2015 Genetic and Evolutionary Computation Conf., pages 1085–1092.
[30] A. Ramírez, J.R. Romero, S. Ventura, A comparative study of many-objective evolutionary algorithms for the discovery of software architectures, Empir. Softw. Eng. 21 (6) (2016) 2546–2600.
[31] G.R. Santhanam, Qualitative optimization in software engineering: a short survey, J. Syst. Softw. 111 (2016) 149–156.
[32] C. Simons, J. Singer, D.R. White, Search-based refactoring: metrics are not enough, 7th Int. Symp. Search-Based Software Eng. (2015) 47–61.
[33] C.L. Simons, I.C. Parmee, Elegant object-oriented software design via interactive, evolutionary computation, IEEE Trans. Syst., Man, Cybern. Part C 42 (6) (2012) 1797–1805.
[34] C.L. Simons, J. Smith, P. White, Interactive ant colony optimization (iACO) for early lifecycle software design, Swarm Intell. 8 (2) (2014) 139–157.
[35] C. Szyperski, Component Software: Beyond Object-Oriented Programming, 2nd edition, Addison-Wesley Longman, Boston, MA, 2002.
[36] H. Takagi, Interactive evolutionary computation: fusion of the capabilities of EC optimization and human evaluation, Proc. IEEE 89 (9) (2001) 1275–1296.
[37] S. Vathsavayi, Hadaytullah, K. Koskimies, Interleaving human and search-based software architecture design, Proc. Estonian Acad. Sci. 62 (1) (2013) 16–26.
[38] A. Vescan, An evolutionary multiobjective approach for the dynamic multilevel component selection problem, in: Int. Conf. on Service-Oriented Computing, pages 193–204, Springer, Berlin Heidelberg, 2016.

7

Other publications associated to this Ph.D. Thesis

7.1. Evolutionary composition of QoS-aware web services: a many-objective perspective



Title	Evolutionary composition of QoS-aware web services: a many-objective perspective
Authors	A. Ramírez, J.A. Parejo, J.R. Romero S. Segura, A. Ruiz-Cortés
Journal	Expert Systems with Applications
Volume	72
Pages	357-370
Year	2017
Editorial	Elsevier
DOI	10.1016/j.eswa.2016.10.047

IF (JCR 2017)	3.768
Categories	Computer Science, Artificial Intelligence Operations Research and Management Science
Positions	20/132 (Q1), 8/83 (Q1)
Cites	3 (WoS), 4 (Scopus)

Expert Systems With Applications 72 (2017) 357–370

Contents lists available at ScienceDirect

Expert Systems With Applications

journal homepage: www.elsevier.com/locate/eswa

Evolutionary composition of QoS-aware web services: A many-objective perspective

Aurora Ramírez [a], José Antonio Parejo [b], José Raúl Romero [a,*], Sergio Segura [b], Antonio Ruiz-Cortés [b]

[a] Department of Computer Science and Numerical Analysis, University of Córdoba, Campus de Rabanales, 14071, Córdoba, Spain
[b] Department of Computing Languages and Systems, University of Sevilla, ETSII, Avda. de la Reina Mercedes, s/n, 41012, Sevilla, Spain

ARTICLE INFO

Article history:
Received 5 November 2015
Revised 14 October 2016
Accepted 20 October 2016
Available online 29 October 2016

Keywords:
QoS-aware web service composition
Many-objective evolutionary algorithms
Multi-objective optimization

ABSTRACT

Web service based applications often invoke services provided by third-parties in their workflow. The Quality of Service (QoS) provided by the invoked supplier can be expressed in terms of the Service Level Agreement specifying the values contracted for particular aspects like cost or throughput, among others. In this scenario, intelligent systems can support the engineer to scrutinise the service market in order to select those candidates that best fit with the expected composition focusing on different QoS aspects. This search problem, also known as QoS-aware web service composition, is characterised by the presence of many diverse QoS properties to be simultaneously optimised from a multi-objective perspective. Nevertheless, as the number of QoS properties considered during the design phase increases and a larger number of decision factors come into play, it becomes more difficult to find the most suitable candidate solutions, so more sophisticated techniques are required to explore and return diverse, competitive alternatives. With this aim, this paper explores the suitability of many-objective evolutionary algorithms for addressing the binding problem of web services on the basis of a real-world benchmark with 9 QoS properties. A complete comparative study demonstrates that these techniques, never before applied to this problem, can achieve a better trade-off between all the QoS properties, or even promote specific QoS properties while keeping high values for the rest. In addition, this search process can be performed within a reasonable computational cost, enabling its adoption by intelligent and decision-support systems in the field of service oriented computation.

1. Introduction

Current service oriented applications need to integrate third-party services, like authentication or persistent storage, as part of their core features. This integration clearly benefits code reuse and modularity, though it also introduces additional concerns. In this sense, the Quality of Service (QoS) experienced by end-users will also depend on the QoS provided by these external services. For example, if the persistence service is unavailable, the performance of the entire application would probably drop or even the system itself would become useless. In such a scenario, a careful selection of the services to be integrated is critical to determine the composition that better achieves an appropriate overall QoS at affordable cost. Moreover, the Service Level Agreement (SLA), *i.e.* the piece of a service contract where the level of service is determined, could even bring more alternatives into play, considering that a specific service provider might offer several QoS configurations for the same service provision. For example, Amazon Web Services (AWS) establishes up to 8 different deployment plans (instances) for their computing services (EC2) (Wada, Suzuki, Yamano, & Oba, 2012), which combined with other configurable options like the operating system, the available CPU and backup settings can lead to 16,991 possible configurations (García-Galán, Rana, Trinidad, & Cortés, 2013). Although highly demanded because of its flexibility, considering a larger set of configuration alternatives implies increasing the number of QoS properties coming into play and, consequently, finding appropriate trade-offs among them becomes extremely difficult. For instance, notice that an investment in CPU and exclusive dedication would improve response time but increasing the cost.

Therefore, deciding which are the most appropriate services to be included in an application, or their specific QoS configurations, is a challenging task for designers, the automatic support acquiring

* Corresponding author. Phone: +34957212660. Fax: +34957218360.
E-mail addresses: aramirez@uco.es (A. Ramírez), japarejo@us.es (J.A. Parejo), jrromero@uco.es (J.R. Romero), sergiosegura@us.es (S. Segura), aruiz@us.es (A. Ruiz-Cortés).

http://dx.doi.org/10.1016/j.eswa.2016.10.047
0957-4174/

even more relevance as the number of decision factors increases. Here, intelligent systems may help to support them by applying search techniques to the exploration and selection of design alternatives. Consequently, analysing how different search methods behave and how they are influenced by the problem structure, e.g. different number of services being orchestrated in different ways, becomes important for these systems to gain efficiency and effectiveness.

The so-called QoS-aware Web Service Composition (QoSWSC) problem has been identified as a key research problem in the service oriented computing (SOC) field (Papazoglou, Traverso, Dustdar, & Leymann, 2007), which is actually NP-hard (Ardagna & Pernici, 2007; Bonatti & Festa, 2005). Although this problem was originally formulated as a single-objective optimisation problem, notice that multiple, often conflicting QoS properties need to be simultaneously considered to address this problem (Wada et al., 2012; Zeng et al., 2004). For example, availability, response time, throughput or invocation cost can be clearly opposed, *e.g.* improving the availability of a service probably would imply an increase in cost. In general, identifying priorities among these attributes is not straightforward, and designers have to ensure an appropriate trade-off between all of them according to their interests.

Given the large number of alternatives to be analysed, a computational optimisation approach can serve to efficiently find the best orchestration of candidate services (Canfora, Penta, Esposito, & Villani, 2005). Both single-objective and multi-objective evolutionary algorithms (MOEAs) constitute a commonly used alternative in the literature, where the latter return a set of solutions, each one achieving a different trade-off between all the objectives (Coello Coello, Lamont, & Van Veldhuizen, 2007). Actually, the use of multi-objective optimisation algorithms has proved to be a more convenient approach to deal with the QoSWSC problem, as recently discussed by Moustafa and Zhang (2013); Suciu, Pallez, Cremene, and Dumitrescu (2013); Trummer, Faltings, and Binder (2014); Wada et al. (2012); Yu, Ma, and Zhang (2015). Having a set of alternative solutions to choose among facilitates better comprehension of the different possible trade-offs between the QoS properties involved in the decision-making process. In contrast, this information would not be available in advance when a single-objective evolutionary approach is applied using a weighted sum.

The application of multi-objective evolutionary approaches to the QoSWSC problem has been mostly focused on the selection of well-known approaches, like NSGA-II (Nondominated Sorting Genetic Algorithm II) or MOGA (Multi-Objective Genetic Algorithm), and considering up to 5 QoS properties as optimisation objectives. Nevertheless, when a large number of objectives need to be considered, the performance of these classical MOEAs tends to drop off as the complexity of the resulting optimisation problem increases. This factor has led to the appearance of new specific approaches, like many-objective evolutionary algorithms, which have emerged as an effective alternative to efficiently explore highly dimensional objective spaces (Ishibuchi, Tsukamoto, & Nojima, 2008). Similarly, many-objective evolutionary algorithms operate in accordance to the precepts of the multi-objective approach. In fact, for situations where engineers need to deal with a large number of decision criteria, as during the design of complex web service compositions considering multiple QoS properties, many-objective optimisation provides an excellent support mechanism.

In this paper, the QoSWSC problem is addressed from the emerging many-objective perspective considering a large number of QoS properties. More specifically, our research question can be phrased as follows: *Is the application of many-objective algorithms appropriate to address in a generalisable way the QoSWSC problem considering a diversity and large number of QoS properties?* To accurately respond to this question, a comparative study of different evolutionary algorithms is proposed with the aim of analysing their suitability when 9 diverse QoS properties constrain the problem statement. Notice that the jointly optimisation of a large number of objectives constitutes a real challenge to any optimisation approach. It could be expected that these algorithms, primarily conceived to deal with such a complexity, will provide a better performance than that obtained by long-standing MOEAs in terms of both the obtained QoS values and the expected balance among then. Finally, we include an in-depth discussion of the empirical insights obtained from the most fitting algorithm in terms of a representative subset of properties, such as runtime and design-time properties. The experimentation with many-objective approaches provides valuable information about how robust and effective these search methods are, which brings the opportunity to incorporate them into intelligent systems aimed at supporting the resolution of more realistic formulations of the QoSWSC problem.

The rest of the paper is organised as follows. Section 2 introduces the multi-objective evolutionary optimisation, as well as the bases for the QoSWSC problem as an optimisation problem. Section 3 describes the related work and then Section 4 explains the specific features of the implemented evolutionary approach, including the definition of the QoS properties considered to evaluate how objectives are met. A detailed performance analysis of the algorithms is conducted in Section 5, where findings and outcomes are also discussed. Then, the threats to validity concerning the presented study are detailed in Section 6. Finally, Section 7 outlines some concluding remarks.

2. Background

In this section, the key concepts of multi- and many-objective optimisation are introduced. Next, the QoS-aware binding of web services is presented as a search problem.

2.1. Multi- and many-objective evolutionary algorithms

Multi-objective evolutionary algorithms are population-based metaheuristics devoted to solve multi-objective optimisation problems (MOPs) (Coello Coello et al., 2007). Just like evolutionary algorithms (EAs), MOEAs define a set of candidate solutions, *i.e.* the population of individuals, which are modified through some iterations seeking for the generation of better solutions. Usually, each solution, also called individual, is encoded using a fixed-length numerical array, *i.e.* the genotype. Continuing the simile, the phenotype is defined as the real-world representation of the genotype.

After the random creation of the initial population, the algorithm starts an iterative process, as shown in Algorithm 1. In every generation, some individuals are selected to act as parents, and genetic operators, like crossover and mutation, are applied to cre-

Algorithm 1 Pseudocode of an evolutionary algorithm.

Require: *maxGenerations*, *populationSize*
1: $population \leftarrow$ createPopulation($populationSize$)
2: evaluate($population$)
3: $generation \leftarrow 0$
4: **while** $generation <= maxGenerations$ **do**
5: $parents \leftarrow$ selection($population$)
6: $offspring \leftarrow$ crossover($parents$)
7: $offspring \leftarrow$ mutation($offspring$)
8: evaluate($offspring$)
9: $population \leftarrow$ replacement($population \cup offspring$)
10: $generation++$
11: **end while**

ate new solutions. On the one hand, the crossover operator recombines genetic information of two parents, resulting in one or more descendants (also known as offspring) that tend to be similar to their parents. Therefore, this operator is expected to promote convergence in the search process. On the other hand, the mutation phase produces some alterations on a given offspring, *e.g.* changing one value in the genotype, with the aim of introducing diversity in the population. The generation ends with the selection of the set of individuals, from both the current population and the offspring pool, that will take part in the next population. Frequently, this replacement mechanism tries to promote the survival of the best solutions found so far, ensuring that diversity is also preserved. The evolutionary process continues until a stopping criterion, *e.g.* a maximum number of generations, is reached.

The main difference between EAs and MOEAs lies on the evaluation of individuals, which determines "how good" a solution is in solving the optimisation problem. In EAs, a unique fitness function is defined to measure the quality of the solutions, so individuals can be directly compared using this fitness value. Instead of defining a specific set of weights to aggregate several objectives, a multi-objective approach treats each objective as an independent function. Not only the problem of determining the weight to be assigned to each function is removed, but also the limitations offered by the use of aggregation functions (Deb, 2001) are overcome. Among others, the use of such a scalarisation function assumes the linearity of the MOP and non-convexity of the PF. Besides, it is not possible to assure that there would be a one-to-one correspondence between the solution optimising the weighted sum and a supposedly non-dominated solution.

Given that MOPs are characterised by the presence of 2 or more objectives, each evaluated solution contains a set of objective values. As for their comparison, the Pareto dominance concept is frequently included as a discerning criterion to choose between two solutions, a and b, which is defined as follows: a is said to dominate b, if and only if a is better or equal than b for all the objectives, and better for at least one objective than b. If this condition is not satisfied, individuals are referred as equivalent or non-dominated. Thus, the purpose of any MOEA is to find the set of non-dominated solutions, i.e. the Pareto set, establishing different trade-offs among all the objectives. Mapping these solutions onto the objective space allows getting the Pareto front (PF).

Some of the most well-known proposals in the field, like SPEA2 (Strength Pareto Evolutionary Algorithm 2) (Zitzler, Laumanns, & Thiele, 2001) and NSGA-II (Deb, Pratap, Agarwal, & Meyarivan, 2002), are strongly based on the Pareto dominance principle to guide the search towards the PF. On the one hand, SPEA2 assigns a strength value to each individual, i, considering both the number of solutions it dominates and the solutions dominating i. On the other hand, NSGA-II ranks the population by fronts, where each front comprises those equivalent solutions that dominate solutions allocated in the following fronts. Regarding diversity preservation, SPEA2 uses the k-nearest neighbour method to estimate the density at any point of the objective space, whereas NSGA-II proposes a crowding distance to discard between solutions belonging to the same front. Additionally, SPEA2 also defines an archive of solutions with a fixed size, where non-dominated solutions are kept.

Certainly, SPEA2 and NSGA-II have shown a good performance in a variety of problem domains when 2 or 3 objectives are considered. However, real-world applications might require the definition of a greater number of objectives, which has lead to a growing interest in solving the so-called many-objective optimisation problems. Although the actual difference between multi- and many-objective problems has not been clearly stated in the literature (Purshouse & Fleming, 2007), most authors agree today with the idea that many-objective problems require the presence of at least 4 objectives (Deb & Jain, 2014; von Lücken, Barán, & Brizuela, 2014). With the increasing complexity of MOPs, concepts like the Pareto dominance and distances, which characterise the aforementioned algorithms, lose the efficiency required to properly guide the search (Khare, Yao, & Deb, 2003; Praditwong & Yao, 2007), motivating the appearance of more sophisticated techniques. In this sense, advances within the field of many-objective optimisation are mainly focused on the adaptation of the dominance principle, the inclusion of specific diversity preservation mechanisms and the use of quality indicators as key features to control the evolution (Wagner, Beume, & Naujoks, 2007).

For instance, MOEA/D (Multiobjective Evolutionary Algorithm based on Decomposition) (Zhang & Li, 2007) proposes a decomposition approach creating a number of subproblems to be simultaneously optimised. Each subproblem associates a different weight to each objective and, as a result, multiple search directions are explored during the search.

Modifying the classical Pareto dominance also serves to improve the performance of MOEAs, since the percentage of non-dominated solutions in a population rapidly grows when the number of objectives increases (Ishibuchi et al., 2008). ϵ-MOEA (Deb, Mohan, & Mishra, 2003) defines a special type of dominance, called ϵ-dominance, which can be applied on an objective space divided in hypercubes or grids. Thus, solutions are compared considering the hypercubes they belong to, instead of its objective values. Moreover, the evolution process tries to generate an unique solution for each hypercube in favour of diversity, saving them in an archive. Similarly, GrEA (Grid-based Evolutionary Algorithm) (Yang, Li, Liu, & Zheng, 2013) also proposes a landscape partition, though the grids are dynamically created in each generation. The sorting approach defined by NSGA-II is considered, as well as some diversity metrics based on the grids.

Another kind of algorithms are the so-called indicator-based approaches. An indicator allows summarising the quality of the overall PF in a real value (Coello Coello et al., 2007), so it can be used to guide the search process. This idea is explored by IBEA (Indicator-based Evolutionary Algorithm) (Zitzler & Künzli, 2004), which proposes a generic multi-objective evolutionary algorithm where the selected indicator is used in both the selection and the replacement stages. Another interesting approach is HypE (Hypervolume Estimation Algorithm) (Bader & Zitzler, 2011), in which the hypervolume (*HV*) indicator is estimated using Monte Carlo simulations. *HV* is one of the most frequently used measures to evaluate PFs, serving to calculate the hyper-area covered by the Pareto front. Another relevant indicator is spacing (*S*), which computes the diversity of the solutions composing the Pareto set.

Finally, NSGA-III (Deb & Jain, 2014) is a reference-point-based method that modifies the behaviour of NSGA-II regarding its diversity preservation technique. Instead of computing the crowding distance, this algorithm defines a set of well-distributed points that are used to promote the individuals that are close to these points at the replacement step.

2.2. QoS-aware binding of composite web services as an optimisation problem

The QoSWSC problem can be defined as the search for the best subset of candidate services to accomplish a composite service within a specific workflow. More precisely, the set of services requiring binding (henceforth named tasks) is identified. For each task t_i, the set of service providers available $S_i = \{s_{i,1}, \ldots, s_{i,m}\}$ (named candidate services) is determined. This set can be obtained by searching in a service registry, or by analysing the set of QoS configurations available in the SLA of the service. Thus, even when a single provider is available, multiple candidate services could be evaluated (one for each alternative QoS configuration provided by the SLA).

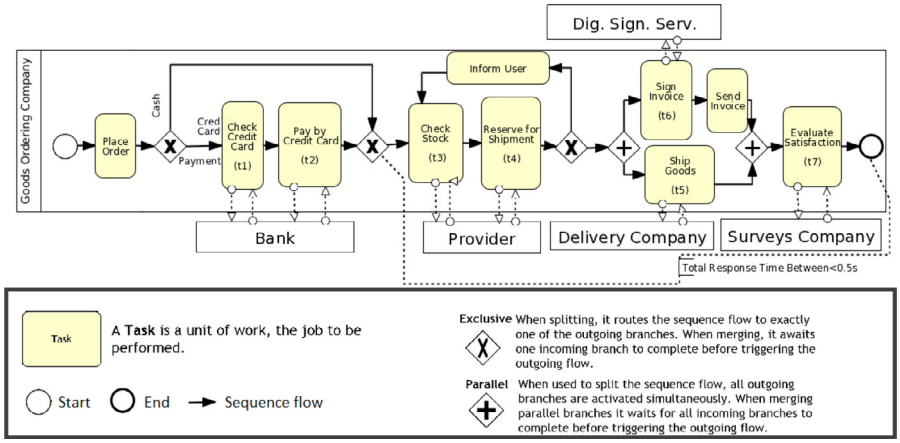


Fig. 1. Goods ordering composite service, adopted from (Parejo et al., 2014).

Table 1
Service providers, candidate services and QoS values for the Goods ordering composite service.

Task and services														
Actor	Bank				Provider				Delivery		Dig. sign.		Surveying	
Provider	*A*		*B*		*C*		*D*		*E*	*F*	*G*	*H*	*I*	*J*
Task	t_1	t_2	t_1	t_2	t_3	t_4	t_3	t_4	t_5	t_5	t_6	t_6	t_7	t_7
Candidate service	$s_{1,A}$	$s_{2,A}$	$s_{1,B}$	$s_{2,B}$	$s_{3,C}$	$s_{4,C}$	$s_{3,D}$	$s_{4,D}$	$s_{5,E}$	$s_{5,F}$	$s_{6,G}$	$s_{6,H}$	$s_{7,I}$	$s_{7,J}$
QoS properties														
Cost (in cents)	1.00	2.00	1.50	5.00	1.00	2.00	1.00	5.00	1.00	2.00	1.00	2.00	1.50	5.00
Exec. time (in seconds)	0.20	0.20	0.10	0.15	0.20	0.20	0.40	0.25	0.20	0.20	0.20	0.20	0.10	0.15

To illustrate this conceptual framework, Fig. 1 shows a goods ordering service using the Business Process Modelling Notation (BPMN). The example presents a business process composed of 7 tasks ($t_1, \ldots, t_7$) with alternative providers including the payment process, the stock management, the delivery and the request of survey questions on the user satisfaction. Note that some of these tasks need to be developed following a specific sequence (*e.g.* t_1 and t_2), whilst others require more complex building blocks. For example, if a product is not available, the application reports about the delay, waiting for some time before repeating t_3 and t_4, *i.e.* a loop will be executed. It is worth noting that the same provider must be chosen for the tasks t_3 and t_4, since the reservation in t_4 refers to the stock of the specific provider queried in t_3. This constraint is denoted in the diagram using the elevation event of BPMN linked to both tasks (an arrow up inscribed in a circle). Next, t_5 and t_6, which belong to two different branches, can be performed in parallel. Finally, the completion of a user satisfaction survey is requested in task t_7.

Once the structure of the composition and its tasks have been defined, a mechanism to choose among candidate services has to be specified. Here, the goal is to find the binding of services (χ) that maximises the global QoS (χ^*) according to the consumers' preferences. The set of QoS properties that need to be satisfied, such as the execution time, availability or cost, among others, is denoted by $\mathbb{Q}$. For each QoS property $q \in \mathbb{Q}$, a global QoS level, Q_q, can be reached from the individual QoS values stipulated by the agreement of each candidate service appearing in χ. Table 1 details the candidate services for the example of Fig. 1 and its QoS values in terms of cost and execution time. For instance, invoking the payment service t_2 of the provider *A*, $s_{2,A}$, costs 0.02$. In this case, notice that the global cost Q_{cost} of a composite web service containing a loop inside will depend on the number of iterations performed.

In order to obtain the set of Q_q values of a specific binding of services, the QoS values of each $s_{i,j}$ in χ are aggregated using a utility function, U_q, where the specific expression to calculate such a function clearly depends on the nature of q. Thus, utility functions express user preferences, *i.e.* the obtained values allow users to decide if a given solution fulfils their expectations or satisfies the existing constraints for a given QoS property. For instance, a total cost of 0.2$ could be fair for some users, but excessive for others.

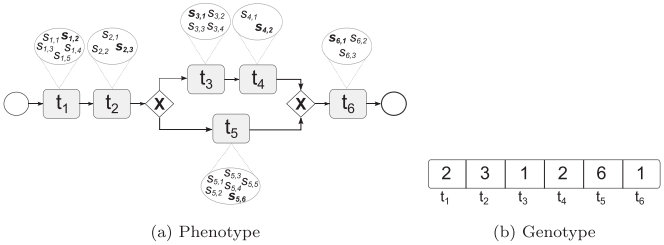
Each utility function U_q also needs to consider the sort of blocks taking part in the composition workflow. The existence of conditional branches entails the possibility that only the services allocated in one branch will be executed, being a different scenario than the invocation of a sequence of them. In this sense, U_{time} for a sequence can be established as the sum of the execution time of all the services that compound that sequence, whilst for a branch the overall value can be defined as the maximum execution time of any path in this branch. However, U_{cost} for a branch is computed as the sum of the cost of tasks in each branch.

Furthermore, specific runtime conditions for loops and alternative branches also influence the calculation of every Q_q. On the one hand, the presence of a loop implies that one service could be invoked many times. On the other hand, not all the services in a conditional structure will be executed in every invocation. In such cases, the total cost will be affected by the choice among alternatives and the number of iterations in the loop, respectively. Since these parameters are unknown in advance, an estimation of the expected behaviour is usually adopted in the literature when defining the problem statement (Canfora, Penta, Esposito, & Villani, 2008). For the goods ordering service example, the average number of iterations per loop could be estimated to include the expected number of querying (t_3) and reservation (t_4) of products in stock. Similarly, the probability of executing each branch of the workflow should be considered. For example, a use of credit card in 80% of payments could be assumed, whilst t_3 and t_4 are executed twice on average before bringing the order to completion. In total, the estimated global cost for a binding $\chi = (A, B, D, D, F, H, J)$ can be

computed as $Q_{Cost}(\chi) = \text{Cost of switch}(\chi) + \text{Cost of Loop}(\chi) + \text{Cost of fork}(\chi) + Cost_7(\chi) = 0.8 * (0.01 + 0.02) + 2 * (0.01 + 0.05) + (0.02 + 0.02) + 0.05 = 0.23\$$.

Since those values are estimations, the actual global QoS values provided can differ significantly in some invocations.

The final assignment of services into tasks is often obtained after a complex decision-making process, being its aim to find the solutions χ^* that maximise the utility of the global QoS values provided by the application. In the motivating example, where only two providers are available for each task, 128 (*i.e.* 2^7) different bindings are possible.



(a) Phenotype (b) Genotype

Fig. 2. Phenotype and genotype of a candidate individual.

3. Related work

The QoSWSC problem was introduced as a single-objective optimisation problem in (Zeng et al., 2004), where integer programming was applied to its resolution. Since then, several non-evolutionary approaches have been used to address this problem. An improved discrete immune optimisation algorithm based on particle swarm optimisation (IDIPSO) has been proposed in (Zhao et al., 2012), whereas in (Parejo, Segura, Fernández, & Ruiz-Cortés, 2014) the authors used GRASP with path relinking to provide appropriate solution to the problems that required a response in short execution time. Those approaches are compiled in (Jula, Sundararajan, & Othman, 2014; Strunk, 2010). The first genetic approach was proposed by (Canfora et al., 2005), also considering a single-objective problem statement.

More recently, the optimisation problem has been addressed from a multi-objective perspective, applying either metaheuristics or other kind of approaches. In (Li & Yan-xiang, 2010) the multi-objective chaos ant colony optimisation algorithm is proposed, showing that it can outperform MOGA under specific conditions when dealing with 3 objectives (*i.e.* cost, time, and reliability). Precisely, MOGA is the basis of the evolutionary framework presented in (Wada et al., 2012), which also considers 3 QoS properties (*i.e.* throughput, latency and cost) to guide the search for Pareto optimal solutions. Reinforcement learning was the selected technique in (Moustafa & Zhang, 2013) to jointly optimise availability, response time and cost, where the experimentation was carried out over a synthetic dataset with only 4 tasks. Particle swarm optimisation (PSO) was also adopted in (Yin, Zhang, Zhang, Guo, & Liu, 2014) to optimise 3 objectives. In this case, sequence and parallel structures constituted the only available building blocks to define the workflow of the composition.

Metaheuristics approaches like EAs, scatter search and PSO were compared to exact methods in (Trummer et al., 2014) in order to optimise up to 5 objectives (*i.e.* response time, availability, throughput, successability, and reliability), extracted from the QWS Dataset (Al-Masri & Mahmoud, 2008). This dataset was also used in (Zhang, 2014) considering all the available QoS properties. In this case, the optimisation problem was solved using PSO, though the experimental study was only performed in terms of the execution time.

Variants of NSGA-II were proposed in (de Campos, Pozo, Vergilio, & Savegnago, 2010) to deal with 5 different objectives, where preference relations were included with the aim of improving the performance of the original algorithm. Finally, MOEA/D was the algorithm selected by (Suciu et al., 2013) to solve a 3-objective variant of the QoSWSC problem. An adaptation of this algorithm using two differential evolution approaches was compared against NSGA-II and GD3 algorithms, also considering different configurations of the number of tasks and candidate services.

A first recent approach using an evolutionary algorithm specifically conceived to deal with many objectives in order to solve the QoSWSC problem is (Yu et al., 2015). In such a paper an adaptation of NSGA-III, named F-MGOP, is proposed. Nevertheless, this work is strictly focused on data-intensive services, and F-MGOP is only compared against SPEA2 and NSGA-II in the empirical validation. In this case, the number of objective functions is limited up to 4 runtime properties (latency, execution cost, availability and accuracy), not usually constituting a real challenge to many-objective algorithms. On the other hand, the study here presented performs a wider and extensive empirical comparison considering diverse algorithms from the different families of many-objective evolutionary approaches and QoS properties of different nature. Furthermore, the conducted analysis (see Section 5) serves to generalise to any service composition the conclusions stated in (Yu et al., 2015) regarding the suitability of many-objective algorithms for optimizing the QoS of data-intensive compositions.

4. Optimisation model

In this section, the common elements of the evolutionary approach are presented, including the encoding of solutions, the genetic operators and the selection and replacement strategies. Evaluation is performed by calculating the objective functions as explained later.

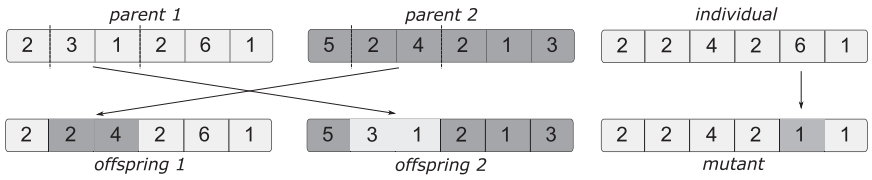
4.1. The evolutionary approach

The genotype/phenotype mapping used in this work follows the approach proposed by (Canfora et al., 2005), and extensively used in the literature (Parejo et al., 2014; Wada et al., 2012). Here, each solution is encoded as an integer array, where each position in the genotype represents a task and its corresponding value, the selected service that will provide it. Consequently, the length of the genotype is equal to the number of tasks appearing in the workflow. Fig. 2 shows the correspondence between the phenotype and the genotype of each individual.

Initialisation. The initialisation of the population is a random procedure, *i.e.* for each task, a web service (highlighted in bold typeface) is randomly chosen from its list of candidate services.

Operators. The adopted genetic operators are those proposed by (Canfora et al., 2005). The *two points crossover* establishes two cut-points in the genotype of the parents, so each descendant is created by recombining one part of one parent and two parts of the other parent. The generated individuals represent two new compositions where the service assigned to each task is inherited from one of its two parents. An example of this procedure is depicted in Fig. 3a. With regard to the mutation procedure, the *one locus mutator* is performed after the crossover by choosing a random gen from the genotype of an offspring in order to change its value. As a result, a new binding is generated for the task associated to this position, as shown in Fig. 3b.

Selection and replacement. Notice that both strategies are defined by each evolutionary algorithm, since they are not domain-specific. Table 2 summarises the characteristics of these procedures for all



(a) Crossover (b) Mutator

Fig. 3. Genetic operators.

Table 2
Selection and replacement strategies defined by each evolutionary algorithm.

Algorithm	Selection	Replacement	Archive update
SPEA2	Comparison of fitness values, combining a strength value and density information	Offspring replace the current population	Updated with non-dominated solutions at each generation
NSGA-II	Tournament selection based on the ranking position (front) and the crowding distance	Progressively stores individuals by fronts, crowding distance takes a decision on the last front	–
MOEA/D	For each individual, two random neighbours generate an offspring	Each offspring is compared against its neighbours	Not used
ϵ-MOEA	An individual within the population is recombined with another one belonging to the archive	The new individual replaces one or more members of the population according to the Pareto dominance	The ϵ-dominance and the already filled hypercubes determine whether the new individual is included or not
GrEA	Tournament selection based on dominance and grid measures	Progressively stores individuals by fronts, grid measures takes a decision on the last front	–
IBEA	Tournament selection based on the indicator (fitness value)	Removes individuals with worst fitness value	–
HypE	Tournament selection based on the *HV* estimation	Progressively stores individuals by fronts, minimum loss of *HV* is considered in the last front	–
NSGA-III	Random selection	Progressively stores individuals by fronts, reference points takes a decision on the last front	–

the algorithms described in Section 2.1. The update mechanism of the external archive is also detailed when required. In this sense, two special considerations have been made regarding MOEA/D and ϵ-MOEA, since both algorithms do not define a maximum size for the archive of solutions. Looking for the fairest scenario, both algorithms have been adapted in order to return as many non-dominated solutions as the other algorithms can manage at each generation, *i.e.* the population size. Thus, MOEA/D will be executed without considering the archive, as recommended by its authors (Zhang & Li, 2007), and the non-dominated solutions will be extracted from the final population. In the case of ϵ-MOEA, if the number of non-dominated solutions is greater than the population size, the archive of solutions will be truncated by a post-processing step using the method proposed by SPEA2. Nevertheless, the final number of solutions would slightly vary from one algorithm to another due to their different ability to explore the search space.

4.2. Objective functions

In the QoSWSC problem, the objective functions are those quality attributes that have to be optimised in order to achieve the best possible global quality of the composite web service. The 9 QoS properties of candidate services defined in the QWS Dataset (Al-Masri & Mahmoud, 2008) have been considered in this work:

- *Response time (T)*. The required time to send a request and receive the response from the service, expressed in milliseconds.
- *Availability (A)*. The ratio (percentage) of successful invocations.
- *Reliability (R)*. A measure of the amount of error messages generated during the service execution, *i.e.* the ratio of error messages to the total messages.
- *Throughput (G)*. The number of invocations to the service per second.
- *Latency (L)*. The time required to respond to a request, expressed in milliseconds.
- *Successability (U)*. The ratio (percentage) of requests that were correctly replied.
- *Compliance (C)*. The ratio (percentage) of conformance with the WSDL (Web Services Description Language) specification proposed by the World Wide Web Consortium.
- *Best practices (B)*. The ratio (percentage) of accomplishment of the WS-I Basic Profile, which establishes a set of requirements to promote interoperability.
- *Documentation (D)*. The ratio (percentage) of description tags of the WSDL used in the service documentation, *e.g.* service name, operation name, etc.

As mentioned in Section 2.2, each candidate service provides its own QoS values, which should be combined in order to obtain the overall QoS value for each property in the composite service. Notice that each specific property covers a different quality aspect of a service. For instance, some properties like *C, B* or *D* may be more related to the design and development process, which may affect the correctness, error handling and maintainability of the system; others like *T* and *L* are strongly influenced by the service provider and the network infrastructure, similarly to *R, A, G* and *U*, which may depend on the runtime conditions and other external factors or operating errors. In most cases, a combination of different factors will determine the choice of the QoS properties to be globally considered.

In order to conduct the evaluation of the quality objectives, an utility function is applied for each QoS property q considering the type of building blocks in the given workflow (see Table 3). For instance, the response time (T) of a sequence of service invocations can be obtained as the sum of the individual T values of these services, whilst a loop containing that sequence should also take

Table 3
Utility functions.

QoS property	*Sequence*	*Loop*	*Branch*	*Fork*
Response time (T)	$\sum_{i=1}^{m} T(a_i)$	$k \cdot \sum_{i=1}^{n} T(a_i)$	$\sum_{i=1}^{m} P_i \cdot T(s_i^b)$	$\min_{i=1}^{p} T(s_i^f)$
Availability (A)	$\prod_{i=1}^{m} A(a_i)$	$(\prod_{i=1}^{n} A(a_i))^k$	$\sum_{i=1}^{m} P_i \cdot A(s_i^b)$	$\prod_{i=1}^{p} A(s_i^f)$
Reliability (R)	$\prod_{i=1}^{m} R(a_i)$	$(\prod_{i=1}^{n} R(a_i))^k$	$\sum_{i=1}^{m} P_i \cdot R(s_i^b)$	$\min_{i=1}^{p} R(s_i^f)$
Throughput (G)	$\min_{i=1}^{m} G(a_i)$	$\min_{i=1}^{m} G(a_i)/k$	$\sum_{i=1}^{m} P_i \cdot G(s_i^b)$	$\min_{i=1}^{p} G(s_i^f)$
Latency (L)	$\sum_{i=1}^{m} L(a_i)$	$k \cdot \sum_{i=1}^{n} L(a_i)$	$\sum_{i=1}^{m} P_i \cdot L(s_i^b)$	$\min_{i=1}^{p} L(s_i^f)$
Successability (U)	$\prod_{i=1}^{m} U(a_i)$	$(\prod_{i=1}^{n} U(a_i))^k$	$\sum_{i=1}^{m} P_i \cdot U(s_i^b)$	$\sum_{i=1}^{p} U_i \cdot U(s_i^f)$
Compliance (C)	$(\sum_{i=1}^{m} C(a_i))/n$	$(\sum_{i=1}^{m} C(a_i))/n$	$\sum_{i=1}^{m} P_i \cdot C(s_i^b)$	$(\sum_{i=1}^{m} C(a_i))/n$
Best practices (B)	$(\sum_{i=1}^{m} B(a_i))/n$	$(\sum_{i=1}^{m} B(a_i))/n$	$\sum_{i=1}^{m} P_i \cdot B(s_i^b)$	$(\sum_{i=1}^{m} B(a_i))/n$
Documentation (D)	$(\sum_{i=1}^{m} D(a_i))/n$	$(\sum_{i=1}^{m} D(a_i))/n$	$(\sum_{i=1}^{m} D(s_i^b))/n$	$(\sum_{i=1}^{p} D(s_i^f))/n$

into account the expected number of iterations, k. P_i stands for the probability of executing the branch i.

It is worth mentioning that the last four functions, *i.e.* U, C, B and D, have been specifically designed for this work as a secondary contribution, whilst the rest were adopted from the literature (Ardagna & Pernici, 2007; Canfora et al., 2005; Strunk, 2010; Wang, Tong, & Thompson, 2007; Zeng et al., 2004). With the exception of T and L, all the properties have to be maximised. Furthermore, for the sake of simplicity, the former have been properly inverted in a preprocessing step, and their aggregation functions have been adapted accordingly.

5. Experimentation

In this section, the proposed experimental framework[1] is detailed. Firstly, the design of the experimentation is motivated by explaining how the obtained results have been validated. Next, the experimentation set-up and algorithm parametrisation are described. The analysis of the results is provided not only from the evolutionary perspective, but also on the degree to which each evolutionary algorithm fits the optimisation of the QoS properties under study. In addition, a more thorough analysis of the approach, including its advantages and limitations, is finally discussed at the end of this section.

5.1. Experimentation rationale

The research methodology applied in this work is an empirical study based on a sequence of controlled experiments, as the standard methodology in optimisation approaches. This methodology enables the isolation and control of the factors that might influence the performance of the algorithms and the quality of the resulting service compositions, thus providing a fair comparison framework. The experimentation has been formulated according to the intrinsic characteristics of the optimisation problem under consideration: (a) the number of tasks and the number of candidate services per task mostly determine the size of search space; (b) the specific QoS values of each candidate are required to compute the global QoS value; and (c) the composition structure of the workflow (*i.e.* nested loops, parallel flows, alternative branches, etc.) determines which utility functions are computed.

Bearing these factors in mind, combining all the different configurations and complexities would imply an extremely large number of executions, leading to an unaffordable combinatorial explosion. Therefore, two representative experiments have been conducted in order to ensure meaningful results and conclusions:

[1] Additional material regarding the problem instances, experimentation results and statistical tests is available for reproducibility from http://www.uco.es/grupos/kdis/sbse/RPRSR15

Table 4
Problem instances generation parameters.

Composition structure parameters	
Max. number of tasks	10, 20, 30, 40, 50
Prob. control flows	0.20
Prob. loops	0.45
Prob. branches	0.45
Prob. flows	0.10
Max. nesting levels	5.00
Runtime flow parameters	
Iterations per loop	Gaussian distribution ($\mu = 5.00$, $\sigma = 1.50$)
Number of branches	2.00
Candidate service parameters	
Candidates per task	Gaussian distribution ($\mu = 5.00$, $\sigma = 2.00$)
Candidates for each task	Randomly chosen from the dataset

Experiment #1 It considers web service compositions having a maximum of 10, 20, 30, 40, or 50 tasks, where each task contains a different set comprised of 1 to 11 candidate services, according to the parametrisation given in Table 4. Their composition is randomly chosen, where a total of 15 problem instances have been generated, *i.e.* 3 instances per each maximum number of tasks, each one associated to a different set of candidate services but sharing the workflow. Thus, this experiment is based on a wide spectrum of problem sizes.

Experiment #2 In order to validate the conclusions drawn from Experiment #1, Experiment #2 should serve to prove that the fixed structure, *i.e.* the workflow, does not have a marked influence on the outcomes. Therefore, 15 different structures of composition were generated for 3 representative instances, *i.e.* 10, 30 and 50 tasks, leading to a total of 45 problem instances. If the relative performance of the algorithms does not change in terms of an statistical significant difference, then it could be inferred that the conclusions are valid under the conditions of Experiment #1. All the problem instances used in these experiments were generated by the instance generator proposed in (Parejo et al., 2014), whose parameters are shown in Table 4.

All the experiments are conducted using the dataset proposed by (Al-Masri & Mahmoud, 2008), which provides the specific QoS values for each candidate service. This dataset has been extensively used in the QoS-aware services computing area (Strunk, 2010) and, particularly, in the recent literature about the QoSWSC problem. It is comprised of QoS values obtained after monitoring 2507 real world and publicly available web services, which contributes to make more realistic experiments within the field.

The results of both experiments have been analysed similarly. Each algorithm has been executed 30 times for each problem in-

Table 5
Parameter set-up.

Common parameters	
Population Size	165
Max. evaluations	33,000
Crossover probability	0.70
Mutation probability	0.10
SPEA2	
Parents selector	Binary tournament
Archive size	165
*k*th neighbour	12
MOEA/D	
Neighbourhood size (τ)	10
Max. replacements (Nr)	4
No. weight vectors	165 (H=3)
GrEA	
Divisions (div)	10
IBEA	
Scaling factor (k)	0.05
HypE	
Sampling points (M)	10,000
NSGA-III	
No. reference points (boundary layer)	165 (p=3)
No. reference points (inside layer)	66 (p=2)

stance considering different random seeds. Then, the hypervolume and spacing indicators have been computed over the returned Pareto fronts, taking average values, to compare the quality of the set of solutions returned by each evolutionary algorithm (see Section 2.1). Both indicators vary in the range [0,1] and should be maximised. Since the hypervolume requires all the objective values to fall into the range [0,1], a post-processing step has to be performed in order to normalise the objective values of all the solutions returned. Next, three non-parametric statistical tests (Arcuri & Briand, 2011; Derrac, García, Molina, & Herrera, 2011) have been executed to assess the differences in the performance of the algorithms in terms of the aforementioned indicators. Firstly, the Friedman test for multiple comparison is carried out, where the null hypothesis, H_0, establishes that all the algorithms perform equally well. This test provides a ranking of algorithms, and a critical value to decide whether H_0 can be rejected at a certain level of significance $(1-\alpha)$. However, this test can only report the existence of significant differences, so a post-procedure has to be applied in order to reveal the sort of these differences. This is the case of the Holm test, where the best algorithm found by the Friedman test is compared against the rest of algorithms.

Additionally, the Cliff's Delta test has been executed to perform pairwise comparisons using an effect-size measurement (Arcuri & Briand, 2011). This method allows classifying the difference between pairs of algorithms as negligible, small, medium or large on the basis of specific thresholds (Romano, Kromrey, Coraggio, & Showronek, 2006).

5.2. Experimental set-up

The optimisation approach as well as the proposed experiments have been coded in Java. The evolutionary algorithms have been implemented using the JCLEC framework (Ramírez, Romero, & Ventura, 2015; Ventura, Romero, Zafra, Delgado, & Hervás, 2007). The experiments were run on a HPC cluster of 8 compute nodes with Rocks cluster 6.1 x64 Linux distribution. Each node comprises two Intel Xeon E5645 CPUs with 6 cores at 2.4 GHz and 24 GB DDR memory.

Table 5 shows the parametrisation of the different evolutionary implementations. Notice that some part of the configuration is common to all the algorithms. Both the population size and the maximum number of evaluations have been fixed after some preliminary experimentation. Crossover and mutation probabilities have been configured in accordance to the values proposed by (Canfora et al., 2005). The rest of parameters have been set following each author's recommendation.

Table 6
Statistical comparison of hypervolume in Experiment #1.

i	Algorithm	Ranking (Friedman)	α/i (Holm)
7	NSGA-III	8.0000	0.0071
6	SPEA2	6.1333	0.0083
5	GrEA	6.0000	0.0100
4	MOEA/D	4.8000	0.0125
3	IBEA	4.2667	0.0167
2	NSGA-II	3.4000	0.0250
1	HypE	2.0000	0.0500
0	**ϵ-MOEA**	**1.4000**	

As for the specific set-up of the algorithms, it should be noted that IBEA applies the ϵ-indicator to guide the search, since the exact computation of hypervolume would be prohibitive. The Tchebycheff approach has been selected as the evaluation mechanism applied by MOEA/D, which allows the presence of objective functions with different scales as happens in this optimisation problem. Moreover, ϵ-MOEA has been modified to internally set the lengths of the hypercubes since they depend on the specific problem instance being solved. Given a workflow composition, the minimum and maximum values that a solution could reach for each QoS property are estimated and used to define 10 hypercubes with equal length, *i.e.* the same number of hypercubes established for GrEA.

5.3. Experiment #1

In this section, the results provided by Experiment #1 are shown and discussed. A comparative study is first presented in terms of the evolutionary performance. Secondly, relevant aspects of the search process regarding the trade-offs reached between QoS properties are discussed. Finally, algorithms are also compared in terms of their execution time.

5.3.1. Results and statistical analysis

After the execution of all the algorithms, the Friedman test determines the ranking position obtained by each algorithm regarding the *HV*, as shown in the third column of Table 6, where ϵ-MOEA is reported to achieve the best ranking. In addition, this test reports a statistics, z, at the specified significance level ($\alpha = 0.01$) in order to determine whether H_0 can be rejected. If the obtained value, which follows a F-Distribution, is greater than a critical threshold, then there exist significant differences among the algorithms. In this case, that condition is satisfied given that the critical value, 2.8272, is lower than the respective statistics, $z = 63.1879$. In order to reveal the sort of those differences, the Holm post-procedure is executed, being its outcomes shown in the fourth column of Table 6. In this case, the test has indicated that the control algorithm (ϵ-MOEA) is better than those algorithms having an α/i value lower than 0.025. Consequently, only HypE and NSGA-II perform equally well than ϵ-MOEA, *i.e.* all of them maintain a good trade-off between convergence and divergence along the search process and, as a result, these algorithms return an equivalent set of high-quality solutions.

Next, as explained in Section 5, the Cliff's Delta test allows analysing the relative performance of pairs of algorithms. Table 7 compiles the results for all the possible pairwise comparisons, each cell showing the estimated difference in terms of *HV* and indicating whether such a value can be considered as negligible (n), small (s), medium (m) or large (l) at the significance level $\alpha = 0.01$. For instance, looking at the fourth row, it can be observed that, even

Table 7
Results of the Cliff's Delta test for hypervolume (n=negligible, s=small, m=medium, l=large) ($\alpha = 0.01$).

Algorithm	SPEA2	NSGA-II	MOEA/D	ϵ-MOEA	GrEA	IBEA	HypE	NSGA-III
SPEA2	–	−0.50 (l)	−0.19 (s)	−0.79 (l)	−0.05 (n)	−0.37 (m)	−0.69 (l)	0.96 (l)
NSGA-II	0.50 (l)	–	0.23 (s)	−0.46 (m)	0.48 (l)	0.16 (s)	−0.28 (s)	0.98 (l)
MOEA/D	0.19 (s)	−0.23 (s)	–	−0.55 (l)	0.13 (n)	−0.13 (n)	−0.48 (l)	0.96 (l)
ϵ-MOEA	0.79 (l)	0.46 (m)	0.55 (l)	–	0.69 (l)	0.58 (l)	0.21 (s)	0.92 (l)
GrEA	0.05 (n)	−0.48 (l)	−0.13 (n)	−0.69 (l)	–	−0.29 (s)	−0.60 (l)	0.96 (l)
IBEA	0.37 (m)	−0.16 (s)	0.13 (n)	−0.58 (l)	0.29 (s)	–	−0.44 (m)	0.95 (l)
HypE	0.69 (l)	0.28 (s)	0.48 (l)	−0.21 (s)	0.60 (l)	0.44 (m)	–	0.93 (l)
NSGA-III	−0.96 (l)	−0.98 (l)	−0.96 (l)	−0.99 (l)	−0.96 (l)	0.95 (l)	−0.99 (l)	–

Table 8
Statistical comparison of spacing in Experiment #1.

i	Algorithm	Ranking (Friedman)	α/i (Holm)
7	IBEA	8.0000	0.0071
6	HypE	6.5333	0.0083
5	GrEA	6.4000	0.0100
4	NSGA-III	4.7333	0.0125
3	ϵ-MOEA	4.0000	0.0167
2	SPEA2	2.7333	0.0250
1	MOEA/D	2.6000	0.0500
0	**NSGA-II**	**1.0000**	

when the previous statistical tests do not reveal a significant difference among ϵ-MOEA, NSGA-II and HypE, the difference between ϵ-MOEA and NSGA-II (0.46) has been classified as medium. On the other hand, the difference between ϵ-MOEA and HypE (0.21) is rather small. In addition, differences among SPEA2, MOEA/D and GrEA are reported as nearly negligible.

Focusing on the spacing indicator, Table 8 shows the results of Friedman and Holm tests. The resulting z value is 202.1765, so strong differences on the performance of the algorithms might be expected ($2.8272 < < z$). Again, the Holm test determines that the control algorithm performs better than those algorithms having an α/i value lower than 0.025. NSGA-II is shown to be able to generate a greater variety of solutions than that provided by most of the many-objective algorithms, even though it has not significant differences with SPEA2 and MOEA/D.

Table 9 compiles the obtained differences among the 8 algorithms in terms of S after performing the Cliff's Delta test. As can be observed, SPEA2, NSGA-II, MOEA/D and ϵ-MOEA clearly outperform the rest of algorithms, since most of the differences are classified as large.

From the obtained results, it is worth noticing that SPEA2 and NSGA-II behave similarly with respect to S, since they reach the first positions in the ranking. However, NSGA-II clearly outperforms SPEA2 in terms of HV. In this sense, NSGA-II has shown good scalability when a high number of objectives is considered, only being overtaken by some specific many-objective approaches like ϵ-MOEA and HypE.

Regarding the many-objective evolutionary algorithms, the decomposition approach proposed by MOEA/D, which is aimed at maintaining diversity during the search, allows obtaining good S values. However, it has also a negative impact on the level of optimisation reached by the solutions, as shown in Table 6. The same behaviour is observed when GrEA and NSGA-III are executed, both of them having problems to properly converge to the PF. Just the opposite situation comes about with IBEA or HypE, since both algorithms return better results for the HV than for S. As a general matter, the joint optimisation of both indicators is a complicated task, and only ϵ-MOEA and NSGA-II have achieved good ranking positions in both cases.

5.3.2. Evolutionary influence on QoS properties

This section discusses the existing relation between algorithms and QoS properties. Table 10 provides the big picture of how good each algorithm is for a QoS property. This has been performed by calculating the average values of each property within the PF and, next, counting the number of times that each algorithm achieves the highest value for each property. All the problem instances were considered. Such values are expressed as percentages, the best value for each QoS property being shown in bold typeface. As can be observed, some specific many-objective approaches like ϵ-MOEA, IBEA and HypE reach the best percentages. Moreover, ϵ-MOEA and HypE have the ability to generate high quality solutions for some specific QoS properties without demoting the trade-off among all of them (see Table 6). Notice that the best set of alternative solutions are provided by HypE and ϵ-MOEA algorithms, what can become a relevant factor when software engineers have not a clear preference on the properties to be optimised.

The separation of QoS properties in runtime (G, A, L, U, R, and T) and design-time (C, B, or D) can provide further insights. For the first case, Table 10 shows that HypE provides the best average values, since it achieves the best average values for 5 out of the 6 runtime properties. Regarding design-time properties, this algorithm also provides the best average values for the standards compliance (C) property, even when for the three properties IBEA has reached the best trade-off. Consequently, if runtime properties are promoted against design-time properties, HypE seems to be the most appropriate choice. Similarly, this algorithm provides the best global trade-off.

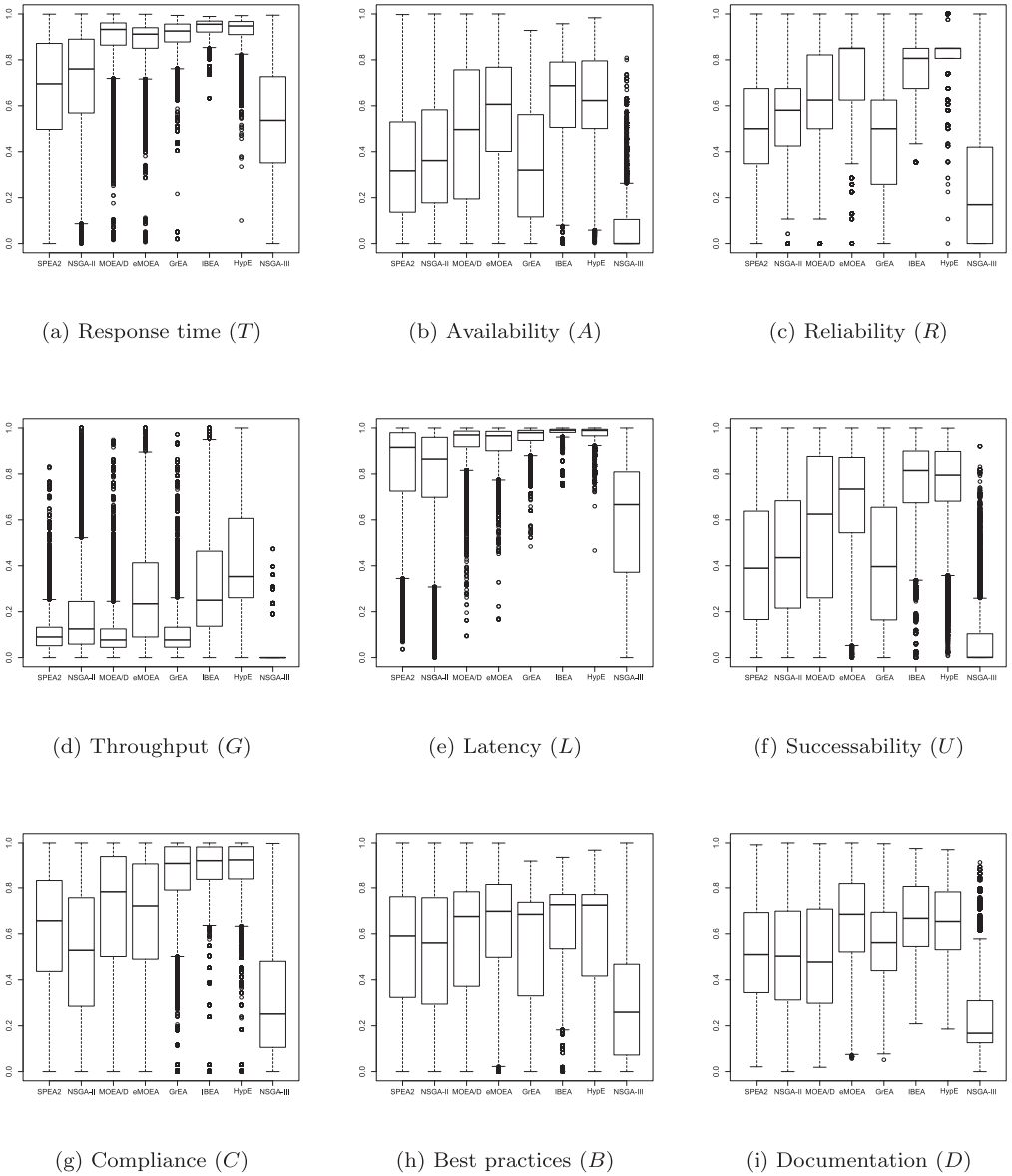
Fig. 4 shows how the QoS values returned by each algorithm are distributed. For the sake of clarity, all the QoS properties are nor-

Table 9
Results of the Cliff's Delta test for spacing (n=negligible, s=small, m=medium, l=large) ($\alpha = 0.01$).

Algorithm	SPEA2	NSGA-II	MOEA/D	ϵ-MOEA	GrEA	IBEA	HypE	NSGA-III
SPEA2	–	−0.88 (l)	−0.07 (n)	0.41 (m)	0.93 (l)	0.93 (l)	0.93 (l)	0.71 (l)
NSGA-II	0.88 (l)	–	0.93 (l)	0.96 (m)	0.93 (l)	0.93 (l)	0.93 (l)	0.93 (l)
MOEA/D	−0.07 (n)	−0.93 (l)	–	0.54 (l)	0.93 (l)	0.93 (l)	0.93 (l)	0.80 (l)
ϵ-MOEA	−0.41 (m)	−0.96 (l)	−0.54 (l)	–	0.96 (l)	0.93 (l)	0.98 (l)	0.27 (s)
GrEA	−1.00 (l)	−1.00 (l)	−1.00 (l)	−0.96 (l)	–	0.95 (l)	0.02 (n)	−1.00 (l)
IBEA	−1.00 (l)	−1.00 (l)	−1.00 (l)	−1.00 (l)	−0.95 (l)	–	−0.93 (l)	−1.00 (l)
HypE	−1.00 (l)	−1.00 (l)	−1.00 (l)	−98 (l)	−0.02 (n)	0.93 (l)	–	−0.99 (l)
NSGA-III	−0.71 (l)	−1.00 (l)	−0.80 (l)	−0.27 (s)	0.93 (l)	0.93 (l)	−0.93 (l)	–

Table 10
Experiment #1: Best algorithms for each QoS property (expressed as percentages).

QoS Property	SPEA2	NSGA-II	MOEA/D	ϵ-MOEA	GrEA	IBEA	HypE	NSGA-III
Response time (T)	0.00	0.00	0.00	0.00	6.67	**53.33**	40.00	0.00
Availability (A)	0.00	6.67	0.00	13.33	0.00	**40.00**	**40.00**	0.00
Reliability (R)	0.00	0.00	0.00	13.33	0.00	6.67	**80.00**	0.00
Throughput (G)	0.00	0.00	0.00	13.33	0.00	6.67	**80.00**	0.00
Latency (L)	0.00	0.00	0.00	0.00	0.00	33.33	**66.67**	0.00
Successability (U)	0.00	0.00	0.00	6.67	0.00	40.00	**53.33**	0.00
Compliance (C)	0.00	0.00	0.00	6.67	13.33	26.67	**53.33**	0.00
Best practices (B)	13.33	0.00	6.67	**40.00**	0.00	20.00	20.00	0.00
Documentation (D)	0.00	0.00	0.00	33.33	6.67	**40.00**	20.00	0.00



(a) Response time (T) (b) Availability (A) (c) Reliability (R)

(d) Throughput (G) (e) Latency (L) (f) Successability (U)

(g) Compliance (C) (h) Best practices (B) (i) Documentation (D)

Fig. 4. Box plots of the distribution of QoS values in the Pareto front found by each algorithm.

malised, and have to be maximised. Notice that differences among algorithms become more distinct. Firstly, ϵ-MOEA, IBEA and HypE obtain not only similar distributions for design-properties, but also a good balance among the rest of attributes. Secondly, NSGA-II provides a wide range of QoS values for all the properties, even when the specific values are lower than those obtained by the aforementioned approaches. In addition, it can be observed that some QoS properties are easier to optimise than others. For instance, latency (L) is highly optimised by most of the algorithms, values greater than 0.8 being frequently achieved. On the contrary, differences between the algorithms are more noticeable for availability (A) and reliability (R).

5.3.3. Analysis of computational cost

As the number of QoS properties increases and the use of more sophisticated algorithms becomes more necessary, it is im-

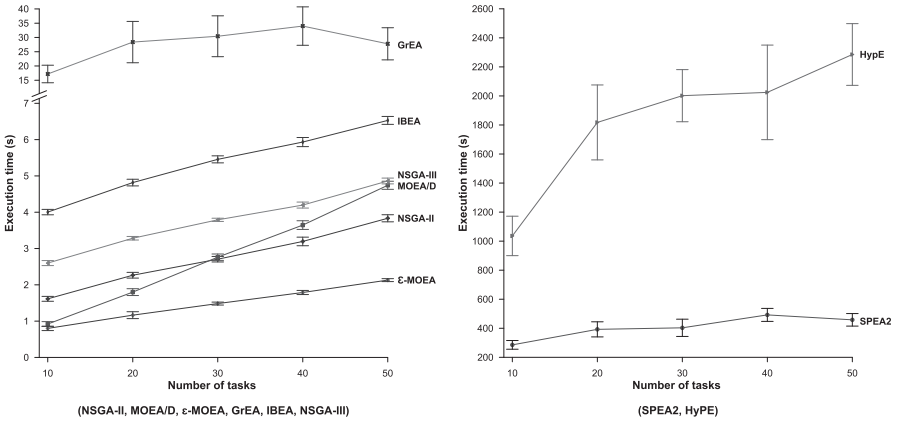


Fig. 5. Average execution time relative to the number of tasks.

Table 11
Statistical comparison of hypervolume in Experiment #2.

i	*Algorithm*	*Ranking (Friedman)*	α/i *(Holm)*
7	NSGA-III	8.0000	0.0071
6	SPEA2	6.4222	0.0083
5	GrEA	5.7778	0.0100
4	IBEA	4.6667	0.0125
3	MOEA/D	4.6444	0.0167
2	NSGA-II	2.9556	0.0250
1	HypE	1.9556	0.0500
0	**ϵ-MOEA**	**1.5778**	

Table 12
Statistical comparison of spacing in Experiment #2.

i	*Algorithm*	*Ranking (Friedman)*	α/i *(Holm)*
7	IBEA	8.0000	0.0071
6	HypE	6.6222	0.0083
5	GrEA	6.3333	0.0100
4	NSGA-III	4.3333	0.0125
3	ϵ-MOEA	3.9778	0.0167
2	SPEA2	4.3463	0.0250
1	MOEA/D	2.4889	0.0500
0	**NSGA-II**	**1.0000**	

portant to confirm that the execution time is suitable to perform the decision-making process. Even though this approach is framed within the design phase, where requirements related to execution time can be met more flexibly, an excessive computational cost could still limit the general adoption of many-objective evolutionary algorithms. Fig. 5 shows the average execution time for all the problem instances used in this experiment, depicting the scalability of the algorithms with respect to the number of tasks. Error bars represent the standard deviation. As can be seen, most of the algorithms require just a few seconds to compute the overall Pareto front, a soft linear increase being observed as the number of tasks grows. Only SPEA2 and, especially, HypE require several minutes to find all the solutions comprising the Pareto front. The software engineer should consider whether such time level is manageable for the project conditions. In addition, it should be noted that obtaining high quality solutions is independent of the execution time required by each specific algorithm. For instance, NSGA-II and ϵ-MOEA are reported as efficient algorithms while they provide the best set of solutions according to the experiment conducted in Section 5.3.1.

5.4. Experiment #2

This experiment serves to analyse the influence of the composition structure on the evolutionary performance or on the optimisation of the QoS properties. Experimentation is performed similarly to Experiment #1.

Table 11 shows the comparison of algorithms in terms of the hypervolume indicator, and reveals the outcomes of the Friedman test considering all the problem instances generated for Experiment #2. As can be observed, the ranking positions for the algorithms are the same that those obtained in Experiment #1, except for IBEA. In this case, z is equal to 220.9533, whereas the critical value is 2.6977. Consequently, since $2.6977 < z$, it can be concluded that there exist significant differences between the algorithms, and the threshold given by the Holm test, 0.05, indicates that ϵ-MOEA is statistically better than the rest of algorithms, except for HypE.

The statistical tests have been computed for the spacing indicator as well (see Table 12). In this case, z is equal to 453.6330, whereas the critical value remains the same. Again, significant differences are revealed after executing the Holm test, which rejects all the hypothesis when comparing the control algorithm, NSGA-II, against the rest of evolutionary approaches.

As previously performed in Experiment #1, Table 13 shows the existing relation between optimisation algorithms and QoS properties. Notice that ϵ-MOEA generates the best solutions in terms of documentation (D) and best practices (B), whereas IBEA seems to promote solutions with good values of standards compliance (C). Again, HypE is mostly focused on the search of solutions that satisfy the 6 runtime properties.

As can be observed, the obtained results remain rather similar to those discussed above. IBEA is likely to be the only exception to this statement, as in Experiment #1 had reach the best position to deal with design-time properties. More specifically, in this case IBEA performs worse regarding availability (A) and documentation (D) and, even when it behaves much better in terms of standards compliance (C), it is outperformed by ϵ-MOEA and HypE globally for the design-time properties. Between the latter approaches, ϵ-MOEA tends to demote the standards compliance (C), whereas HypE reaches a better balance among all the QoS properties. Nevertheless, notice that the three algorithms behave very similarly for the three design-time properties according to the distribution of QoS values (see Fig. 4), which implies that any minor change in the selected problem instances could modify their ranking positions.

With regard to the composite structure, it does not affect the relative performance of algorithms from an evolutionary perspective, as can be observed from the results. In global terms, the ranking positions remain the same, and the observed strengths and weaknesses of each evolutionary approach to explore the search

Table 13
Experiment #2: Best algorithms for each QoS property (expressed as percentages).

QoS Property	SPEA2	NSGA-II	MOEA/D	ϵ-MOEA	GrEA	IBEA	HypE	NSGA-III
Response time (T)	0.00	0.00	0.00	0.00	4.44	46.67	**48.89**	0.00
Availability (A)	2.22	2.22	0.00	13.33	0.00	24.44	**57.78**	0.00
Reliability (R)	0.00	0.00	2.22	6.67	0.00	15.56	**75.56**	0.00
Throughput (G)	0.00	0.00	0.00	4.44	2.22	13.33	**80.00**	0.00
Latency (L)	0.00	0.00	0.00	0.00	2.22	33.33	**64.44**	0.00
Successability (U)	2.22	2.22	0.00	13.33	0.00	17.78	**64.44**	0.00
Compliance (C)	0.00	0.00	0.00	2.22	8.89	**57.78**	31.11	0.00
Best practices (B)	6.67	2.22	0.00	**44.44**	8.89	17.78	20.00	0.00
Documentation (D)	4.44	0.00	0.00	**40.00**	6.67	20.00	28.89	0.00

space are due to other characteristics of the QoSWSC problem, i.e. its highly combinatorial nature and the number of objectives. The observed differences between both experiments can be explained by the fact that Experiment #2 considers a greater number of problem instances, and the addition of new workflow structures. This might influence the returned QoS values and, consequently, the results of the indicators.

5.5. Discussion of results

Understanding the advantages and limitations of the experimental findings can give awareness of the applicability of the proposed approach. Regarding its advantages, the comparative study has provided novel evidences of the performance of six many-objective algorithms to solve the QoSWSC problem, comparing them with two classical multi-objective algorithms. In this sense, results have shown that differences on the evolutionary performance of the algorithms are mostly due to the number of objectives and, consequently, their behaviour is shown to be significantly robust in all cases. On the one hand, many-objective approaches like ϵ-MOEA and HypE have proven to be more effective search methods than multi-objective algorithms in terms of the obtained QoS values. More specifically, experimental results show that ϵ-MOEA provides the best values for the QoS properties being optimised, whereas HypE reaches a better trade-off between the values of the target QoS properties. Even so, they tend to obtain less variety of web service compositions.

An in-depth analysis of the solutions returned by each algorithm points out that some algorithms, such as HypE and IBEA, are able to generate solutions with highly optimised values for some specific QoS properties of runtime (reliability and throughput) and design-time (documentation), respectively. At the same time, they maintain a good balance among the rest of properties. Hence, many-objective approaches become especially well-suited in those cases in which these properties are of particular interest to the engineer. To the best of our knowledge, this kind of study had never been conducted before in the context of the QoSWSC problem. If properly used, this information can be also exploited by the intelligent system aimed at providing the engineer with valuable heuristics for the selection of the most appropriate algorithm for each QoS property.

This proposal has some limitations, too. For instance, some of the algorithms are only suitable to solve the problem at design-time due to their high computational complexity. Nevertheless, notice that their execution time is still affordable at this stage of the development. Even some of the many-objective algorithms applied here, like IBEA, NSGA-III and MOEA/D, are faster than multi-objective approaches like SPEA2. Furthermore, a low execution time does not necessarily conflict with the generation of high quality solutions, as demonstrated by ϵ-MOEA. From the point of view of the decision-making process, the engineer could consider as a drawback the need of selecting a specific solution from the final Pareto front. This is usual when dealing with Pareto-based approaches, which could be configured or adapted to return a smaller set of solutions to choose from.

6. Threats to validity

As any research methodology, the experimental study proposed here presents limitations that should be clearly pointed out. These are described next in terms of internal and external validity threats, including the decisions taken to mitigate their impact.

Internal validity . This refers to whether there is sufficient evidence to support the conclusions and the sources of bias that could compromise those conclusions. In order to minimise the impact of external factors in the obtained results, all the algorithms were executed 30 times per problem instance (market of candidate services and structure of the composition) to compute averages. Moreover, statistical tests were performed to ensure the significance of the differences identified between the results obtained by the compared proposals. Finally, the experiments have been executed in a remote cluster of computers, so a stable experimentation platform was provided.

External validity. This is concerned with how the experiments capture the research objectives and the extent to which the drawn conclusions can be generalised. This can be mainly divided into limitations of the approach and generalisability of the conclusions. Regarding the limitations, experiments did not reveal significant differences for all the pairwise comparisons between algorithms. Nonetheless, the obtained results have provided solid insights when comparing the general behaviour of multi-objective evolutionary algorithms, mostly focused on maintaining diversity, and many-objective approaches, which tend to work better in terms of hypervolume.

Regarding the generalisability of conclusions, the parameters and the size of the analysed problem instances were chosen considering the most common values used in the literature (Strunk, 2010). Additionally, Experiment #2 was performed to ensure the generalisability of the results independently of the specific composition structure used in the problem instances (workflow layout). In this case, up to 45 different problem instances were randomly generated in order to compare the performance of the proposals with disparate composition structures. Since the rankings of the algorithms mostly remain unaltered with respect to Experiment #1, and differences are statistically significant in all cases, it can be concluded that results are generalisable for the composition structures having the applied parameters. Finally, our experimental study does not take into account neither global QoS constraints nor interdependence constraints. Although most of the selected algorithms can be adapted to deal with constrained problems, conclusions regarding their performance cannot be extrapolated from the current study.

7. Concluding remarks

This paper presents a comparative study on the suitability and performance of different multi- and many-objective algorithms to deal with the QoS Web Service Composition problem, which has been identified as a key issue in the field of SOC. This problem has been already addressed from a multi-objective perspective in the near past, when only a small number of properties was under study, *e.g.* cost or availability. Having a few properties leads to an objective space where multi-objective evolutionary algorithms work well. However, in real-world environments these approaches have shown their unsuitability as the number of objectives increases, *e.g.* considering at the same time both runtime and design attributes. Even so, the trade-off among all of them has to be still preserved, and the choice of candidate services becomes a harder task demanding more sophisticated optimisation techniques.

A comparative study of 2 multi-objective and 6 many-objective evolutionary algorithms has been proposed to address a 9-objective (QoS properties) QoSWSC problem, taking into consideration those aspects that the engineer might find in a real environment. This is the first generalisable and extensive application of specific many-objective evolutionary algorithms to solve this optimisation problem, where factors like the number of tasks or the composition structure influence its complexity. Therefore, experiments have also considered a wide range of problem instances using real QoS values.

Experimental results confirm that many-objective algorithms are a suitable option to face QoSWSC problems considering a large number of objectives at design time. Among the implications that can be derived from the conducted analysis, it is worth mentioning the ability of many-objective algorithms like ϵ-MOEA, HypE and IBEA to optimise specific QoS properties. Additionally, the experimental study has shown that the proposed approach is not specifically influenced by the way how the problem is formulated in terms of its structure composition and tasks.

As many-objective optimisation has turned out to be an interesting paradigm to move one step forward in the automatic composition of web services, future research is planned to explore even more complex formulations of the QoSWSC problem. As a next step, adding constraints like service dependencies or the satisfaction of thresholds for certain QoS properties will allow analysing their influence on the search process. In this application domain, it is of particular relevance to study how constraint-handling techniques can be effectively integrated into many-objective evolutionary approaches. Similarly, combining many-objective algorithms with prioritisation techniques would allow focusing the search on those QoS properties of highest interest to the engineer.

In addition, authors plan to explore the possibility of combining the approaches proposed in this paper, aimed at addressing the QoSWSC problem at design time, with other techniques more appropriate to enable the optimisation process at runtime (Parejo et al., 2014). With such a combined approach, the entire life-cycle of the service compositions could be covered, including design-time service selection, optimisation at deployment-time, and runtime reconfiguration. Another important step forward is the application of this approach to a real case study using popular services like Amazon EC2 and PayPal. Finally, we consider relevant for the expert system to let the engineer get involved by the search algorithm using human-in-the-loop models, so that it could explore the search space guided by the experts decisions.

Acknowledgements

Work supported by the Spanish Ministry of Science and Technology and the Andalusian R&I&D, projects P12-TIC-1867, TIN2012-32273, TIC-5906, and FEDER funds. This research was also supported by Spanish Ministry of Economy and Competitiveness, projects TIN2014-55252-P and TIN2015-71841-REDT, and the Spanish Ministry of Education under the FPU program (FPU13/01466).

References

Al-Masri, E., & Mahmoud, Q. H. (2008). Investigating web services on the world wide web. In *Proceedings of the 17th international conference on world wide web*. In *WWW '08* (pp. 795–804). New York, NY, USA: ACM. doi:10.1145/1367497.1367605.

Arcuri, A., & Briand, L. (2011). A practical guide for using statistical tests to assess randomized algorithms in software engineering. In *Proceedings of the 33rd international conference on software engineering*. In *ICSE '11* (pp. 1–10). New York, NY, USA: ACM. doi:10.1145/1985793.1985795.

Ardagna, D., & Pernici, B. (2007). Adaptive service composition in flexible processes. *IEEE Transactions on Software Engineering, 33*(6), 369–384. doi:10.1109/TSE.2007.1011.

Bader, J., & Zitzler, E. (2011). Hype: An algorithm for fast hypervolume-based many-objective optimization. *Evolutionary Computation, 19*(1), 45–76. doi:10.1162/EVCO_a_00009.

Bonatti, P. A., & Festa, P. (2005). On optimal service selection. In *Proceedings of the 14th international conference on world wide web*. In *WWW '05* (pp. 530–538). New York, NY, USA: ACM. doi:10.1145/1060745.1060823.

de Campos, A., Pozo, A., Vergilio, S., & Savegnago, T. (2010). Many-objective evolutionary algorithms in the composition of web services. In *Proceedings of the 11th brazilian symposium on neural networks*. In *SBRN'10* (pp. 152–157). IEEE. doi:10.1109/SBRN.2010.34.

Canfora, G., Penta, M. D., Esposito, R., & Villani, M. L. (2005). An approach for QoS-aware service composition based on genetic algorithms. In *Proceedings of the 7th annual conference on genetic and evolutionary computation*. In *GECCO '05* (pp. 1069–1075). New York, NY, USA: ACM. doi:10.1145/1068009.1068189.

Canfora, G., Penta, M. D., Esposito, R., & Villani, M. L. (2008). A framework for QoS-aware binding and re-binding of composite web services. *Journal of Systems and Software, 81*(10), 1754–1769. doi:10.1016/j.jss.2007.12.792.

Coello Coello, C. A., Lamont, G. B., & Van Veldhuizen, D. A. (2007). *Evolutionary algorithms for solving multi-objective problems* (2nd). Secaucus, NJ, USA: Springer-Verlag New York, Inc.. doi:10.1007/978-0-387-36797-2.

Deb, K. (2001). *Multi-objective optimization using evolutionary algorithms*. New York, NY, USA: John Wiley & Sons, Inc..

Deb, K., & Jain, H. (2014). An evolutionary many-objective optimization algorithm using reference-point-based nondominated sorting approach, part i: Solving problems with box constraints. *IEEE Transactions on Evolutionary Computation, 18*(4), 577–601. doi:10.1109/TEVC.2013.2281535.

Deb, K., Mohan, M., & Mishra, S. (2003). Towards a quick computation of well-spread pareto-optimal solutions. In C. s. Fonseca, P. J. Fleming, E. Zitzler, L. Thiele, & K. Deb (Eds.), *Evolutionary multi-criterion optimization*. In *Lecture Notes in Computer Science: 2632* (pp. 222–236). Springer Berlin Heidelberg. doi:10.1007/3-540-36970-8_16.

Deb, K., Pratap, A., Agarwal, S., & Meyarivan, T. (2002). A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation, 6*(2), 182–197. doi:10.1109/4235.996017.

Derrac, J., García, S., Molina, D., & Herrera, F. (2011). A practical tutorial on the use of nonparametric statistical tests as a methodology for comparing evolutionary and swarm intelligence algorithms. *Swarm and Evolutionary Computation, 1*(1), 3–18. doi:10.1016/j.swevo.2011.02.002.

García-Galán, J., Rana, O., Trinidad, P., & Cortés, A. R. (2013). Migrating to the cloud - a software product line based analysis. In *Proceedings of the 3rd international conference on cloud computing and services science*. In *CLOSER 2013* (pp. 416–426). doi:10.5220/0004357104160426.

Ishibuchi, H., Tsukamoto, N., & Nojima, Y. (2008). Evolutionary many-objective optimization: a short review. In *Proceedings of the ieee congress on evolutionary computation*. In *CEC 2008* (pp. 2419–2426). doi:10.1109/CEC.2008.4631121.

Jula, A., Sundararajan, E., & Othman, Z. (2014). Cloud computing service composition: A systematic literature review. *Expert Systems with Applications, 41*(8), 3809–3824. doi:10.1016/j.eswa.2013.12.017.

Khare, V., Yao, X., & Deb, K. (2003). Performance scaling of multi-objective evolutionary algorithms. In C. M. Fonseca, P. J. Fleming, E. Zitzler, L. Thiele, & K. Deb (Eds.), *Proceedings of the 2nd international conference on evolutionary multi-criterion optimization*. In *Lecture Notes in Computer Science: 2632* (pp. 376–390). Berlin, Heidelberg: Springer. doi:10.1007/3-540-36970-8_27.

Li, W., & Yan-xiang, H. (2010). A web service composition algorithm based on global QoS optimizing with MOCACO. In C.-H. Hsu, L. T. Yang, J. H. Park, & S.-S. Yeo (Eds.), *Algorithms and architectures for parallel processing*. In *Lecture Notes in Computer Science: 6082* (pp. 218–224). Springer Berlin Heidelberg. doi:10.1007/978-3-642-13136-3_22.

von Lücken, C., Barán, B., & Brizuela, C. (2014). A survey on multi-objective evolutionary algorithms for many-objective problems. *Computational Optimization and Applications, 58*(3), 707–756. doi:10.1007/s10589-014-9644-1.

Moustafa, A., & Zhang, M. (2013). Multi-objective service composition using reinforcement learning. In S. Basu, C. Pautasso, L. Zhang, & X. Fu (Eds.), *Service-oriented computing*. In *Lecture Notes in Computer Science: 8274* (pp. 298–312). Springer Berlin Heidelberg. doi:10.1007/978-3-642-45005-1_21.

Papazoglou, M. P., Traverso, P., Dustdar, S., & Leymann, F. (2007). Service-oriented computing: State of the art and research challenges. *IEEE Computer, 40*(11), 38–45. doi:10.1109/MC.2007.400.

Parejo, J. A., Segura, S., Fernández, P., & Ruiz-Cortés, A. (2014). QoS-aware web services composition using GRASP with path relinking. *Expert Systems with Applications, 41*(9), 4211–4223. doi:10.1016/j.eswa.2013.12.036.

Praditwong, K., & Yao, X. (2007). How well do multi-objective evolutionary algorithms scale to large problems. In *Proceedings of the ieee congress on evolutionary computation*. In *CEC 2007* (pp. 3959–3966). IEEE. doi:10.1109/CEC.2007.4424987.

Purshouse, R., & Fleming, P. (2007). On the evolutionary optimization of many conflicting objectives. *IEEE Transactions on Evolutionary Computation, 11*(6), 770–784. doi:10.1109/TEVC.2007.910138.

Ramírez, A., Romero, J. R., & Ventura, S. (2015). An extensible JCLEC-based solution for the implementation of multi-objective evolutionary algorithms. In *Proceedings of the companion publication of the 2015 annual conference on genetic and evolutionary computation*. In *GECCO Companion '15* (pp. 1085–1092). New York, NY, USA: ACM. doi:10.1145/2739482.2768461.

Romano, J., Kromrey, J. D., Coraggio, J., & Showronek, J. (2006). Appropriate statistics for ordinal level data: Should we really be using *t*-test and cohen's d for evaluating group differences on the NSSE and other surveys? In *Annual meeting of the florida association of institutional research* (pp. 1–33).

Strunk, A. (2010). QoS-aware service composition: a survey. In *Proceedings of the 2010 ieee 8th european conference on web services*. In *ECOWS* (pp. 67–74). IEEE. doi:10.1109/ECOWS.2010.16.

Suciu, M., Pallez, D., Cremene, M., & Dumitrescu, D. (2013). Adaptive MOEA/d for QoS-based web service composition. In M. Middendorf, & C. Blum (Eds.), *Evolutionary computation in combinatorial optimization*. In *Lecture Notes in Computer Science: 7832* (pp. 73–84). Springer Berlin Heidelberg. doi:10.1007/978-3-642-37198-1_7.

Trummer, I., Faltings, B., & Binder, W. (2014). Multi-objective quality-driven service selection - a fully polynomial time approximation scheme. *IEEE Transactions on Software Engineering, 40*(2), 167–191. doi:10.1109/TSE.2013.61.

Ventura, S., Romero, C., Zafra, A., Delgado, J. A., & Hervás, C. (2007). JCLEC: A java framework for evolutionary computation. *Soft Computing, 12*(4), 381–392. doi:10.1007/s00500-007-0172-0.

Wada, H., Suzuki, J., Yamano, Y., & Oba, K. (2012). E3: A multiobjective optimization framework for SLA-aware service composition. *IEEE Transactions on Services Computing, 5*(3), 358–372. doi:10.1109/TSC.2011.6.

Wagner, T., Beume, N., & Naujoks, B. (2007). Pareto-, aggregation-, and indicator-based methods in many-objective optimization. In S. Obayashi, K. Deb, C. Poloni, T. Hiroyasu, & T. Murata (Eds.), *Evolutionary multi-criterion optimization*. In *Lecture Notes in Computer Science: 4403* (pp. 742–756). Springer Berlin Heidelberg. doi:10.1007/978-3-540-70928-2_56.

Wang, H., Tong, P., & Thompson, P. (2007). QoS-based web services selection. In *Proceedings of the ieee international conference on e-business engineering*. In *ICEBE 2007* (pp. 631–637). doi:10.1109/ICEBE.2007.109.

Yang, S., Li, M., Liu, X., & Zheng, J. (2013). A grid-based evolutionary algorithm for many-objective optimization. *IEEE Transactions on Evolutionary Computation, 17*(5), 721–736. doi:10.1109/TEVC.2012.2227145.

Yin, H., Zhang, C., Zhang, B., Guo, Y., & Liu, T. (2014). A hybrid multiobjective discrete particle swarm optimization algorithm for a SLA-aware service composition problem. *Mathematical Problems in Engineering, 2014*, 1–14. doi:10.1155/2014/252934.

Yu, Y., Ma, H., & Zhang, M. (2015). F-MOGP: A novel many-objective evolutionary approach to QoS-aware data intensive web service composition. In *Proceedings of the 2015 ieee congress on evolutionary computation*. In *CEC* (pp. 2843–2850). IEEE. doi:10.1109/CEC.2015.7257242.

Zeng, L., Benatallah, B., Ngu, A., Dumas, M., Kalagnam, J., & Chang, H. (2004). QoS-aware middleware for web services composition. *IEEE Transactions on Software Engineering, 30*(5), 311–327. doi:10.1109/TSE.2004.11.

Zhang, Q., & Li, H. (2007). MOEA/D: A multiobjective evolutionary algorithm based on decomposition. *IEEE Transactions on Evolutionary Computation, 11*(6), 712–731. doi:10.1109/TEVC.2007.892759.

Zhang, T. (2014). QoS-aware web service selection based on particle swarm optimization. *Journal of Networks, 9*(3), 565–570. doi:10.4304/jnw.9.3.565-570.

Zhao, X., Song, B., Huang, P., Wen, Z., Weng, J., & Fan, Y. (2012). An improved discrete immune optimization algorithm based on PSO for QoS-driven web service composition. *Applied Soft Computing, 12*(8), 2208–2216. doi:10.1016/j.asoc.2012.03.040.

Zitzler, E., & Künzli, S. (2004). Indicator-based selection in multiobjective search. In X. Yao, E. K. Burke, J. A. Lozano, J. Smith, J. J. Merelo-Guervós, J. A. Bullinaria, J. E. Rowe, P. Tinŏ, A. Kabán, & H.-P. Schwefel (Eds.), *Parallel problem solving from nature - ppsn viii*. In *Lecture Notes in Computer Science: 3242* (pp. 832–842). Springer Berlin Heidelberg. doi:10.1007/978-3-540-30217-9_84.

Zitzler, E., Laumanns, M., & Thiele, L. (2001). SPEA2: improving the strength pareto evolutionary algorithm. In *Proceedings of the conference on evolutionary methods for design, optimisation and control with applications to industrial problems* (pp. 95–100).

7.2. A systematic literature review of interaction in search-based software engineering

Title	A Systematic Review of Interaction in Search-Based Software Engineering
Authors	A. Ramírez, J.R. Romero, C.L. Simons
Journal	IEEE Transactions on Software Engineering
Year	2018
Editorial	IEEE
DOI	10.1109/TSE.2018.2803055

IF (JCR 2017)	3.331
Category	Computer Science, Software Engineering
Position	6/104 (Q1)
Cites	1 (Scopus)

A Systematic Review of Interaction in Search-Based Software Engineering

Aurora Ramírez, José Raúl Romero, *Member, IEEE,* and Christopher L. Simons

Abstract—Search-Based Software Engineering (SBSE) has been successfully applied to automate a wide range of software development activities. Nevertheless, in those software engineering problems where human evaluation and preference are crucial, such insights have proved difficult to characterize in search, and solutions might not look natural when that is the expectation. In an attempt to address this, an increasing number of researchers have reported the incorporation of the 'human-in-the-loop' during search and interactive SBSE has attracted significant attention recently. However, reported results are fragmented over different development phases, and a great variety of novel interactive approaches and algorithmic techniques have emerged. To better integrate these results, we have performed a systematic literature review of interactive SBSE. From a total of 669 papers, 26 primary studies were identified. To enable their analysis, we formulated a classification scheme focused on four crucial aspects of interactive search, i.e. the problem formulation, search technique, interactive approach, and the empirical framework. Our intention is that the classification scheme affords a methodological approach for interactive SBSE. Lastly, as well as providing a detailed cross analysis, we identify and discuss some open issues and potential future trends for the research community.

Index Terms—Search-Based Software Engineering, Interaction, Systematic Literature Review, Optimization

1 Introduction

The design and development of complex, large-scale software systems can be non-trivial and challenging for the software engineer to perform. In an attempt to assist him/her, formulating software development activities as optimization problems has enabled the application of a range of metaheuristic search approaches. Such search-based software engineering (SBSE) [1] approaches have attracted significant research attention in recent years. Indeed, attempts have been made to support the software engineer in many cognitively challenging development activities by applying SBSE approaches across a range of lifecycle activities, as surveyed by Harman et al. [2].

1.1 Problem description

The application of search-based approaches in support of software engineers raises a number of challenges. For example, it is generally difficult to formulate both a solution representation and a fitness measure of appropriate fidelity to fully reflect the reality of the software engineer's development activity [3]. This can be especially challenging when development decisions involve a range of seemingly unrelated and disparate criteria for solution acceptance where many cannot be explicitly articulated [4], [5]. Also, it is important that the run-time performance of the search approach is satisfactory with respect to support for development. Given the typical dimensionality and scale of search-based problems in the software development lifecycle, this may also be challenging [6] (chapter 9).

In addition, results of automated search approaches should engender the trust and acceptance of software engineers. As has been pointed out previously by Klien et al. [7], the software engineer and any computationally intelligent tool must use common ground to work jointly to agreed goals. Klien et al. suggest that *"to be a team player, an intelligent agent, like a human, must be reasonably predictable and reasonably able to predict others actions"*. Without this common ground, it is possible that software engineers may be reticent to apply probabilistic search approaches they cannot control and that produce solutions that do not look *"human-written"* [8]. In an example of search-based automatic software repair using genetic programming [9], the authors report that *"an additional challenge is establishing the credibility of automated repairs in terms of programmers confidence in them and their understandability"*. It is possible to speculate that because of these challenges, and because SBSE is a relatively recent field of research, SBSE does not yet appear to have provoked broad industrial adoption by software engineers.

To address the challenges of applying search to the realities of software development, as well as search performance and trust, attempts have been made to engage the software engineer by incorporating their participation during search. Exploiting software engineer insight via interaction within search can enrich fitness measure fidelity, and assist search performance by steering the trajectory of search to preferred regions, as well as guiding parameters of the search process itself. The notion of exploiting human interaction in search is not new, however. Early examples of human interaction with computational evolution can be found in Dawkins' biomorphs [10] and Sims' evolutionary virtual creatures [11], while a survey of interactive evolutionary

- *A. Ramirez and J.R. Romero are with the Department of Computer Science and Numerical Analysis, University of Córdoba, 14071, Spain. E-mail: aramirez@uco.es,jrromero@uco.es*
- *Christopher Simons is with the Department of Computer Science and Creative Technologies, University of the West of England, Bristol, BS16 1QY, United Kingdom. Email: chris.simons@uwe.ac.uk*

Manuscript received April 1, 2017; revised August 26, 2025.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSE.2018.2803055, IEEE Transactions on Software Engineering

computation was conducted by Tagaki in 2001 [12]. Later, in 2007, Branke et al. report on research into interactive multi-objective optimization [13].

In essence, notice that any attempt to involve the human in the search process with the aim of adapting the results to his/her preferences can be viewed as a sort of interaction [3]. Different approaches involve the capture of human preference *a priori* to set parameters and constraints for subsequent search, while other approaches allow search to reach termination and, *a posteriori*, present candidate solutions to the human for inspection [14]. However, in defining the scope of the problem description, we exclude *a priori* and *a posteriori* approaches and focus solely on direct human participation in search, which allows the user to intervene more than once and therefore react to intermediate results.

Nevertheless, while artificial intelligence (AI) tools can support optimization or knowledge discovery activities, it is possible that they can also introduce new types of errors. According to Lyell and Coiera [15], *"automation bias (AB) happens when users become overreliant on decision support, which reduces vigilance in information seeking and processing"*. The notions of automation bias and user complacency in automated decision support have been reported previously [16], [17], and experiments conducted by Bahner et al. [18] indicate that the perception of false recommendations is associated with high levels of user complacency. Shackelford [19] reports that an over-reliance on repeated 'human-in-the-loop' evaluation and interaction in search can result in user fatigue, wherein a non-linearity of user focus results in inconsistent search trajectory. Lyell and Coiera [15] suggest that strategies to minimize automation bias might focus on cognitive load reduction.

It can also be challenging to capture subjective evaluation of qualitative factors as a fitness measure in metaheuristic search. Indeed, there are some ill-defined aspects of good software systems quality that cannot be articulated, but nevertheless 'you know it when you see it'. This phenomenon has been referred to as the *"quality without a name"* [20], and is endorsed by the software design patterns community [21]. A distinction between implicit versus explicit user feedback and evaluation is drawn by Aljawadeh et al. in the metaheuristic design pattern 'preference' [14].

1.2 Motivation

Reflecting on SBSE in an overview of approaches to realizing artificial intelligence in software engineering (SE), Harman [22] notes a number of research gaps and challenges for future research. Among these are a need for SBSE to provide insight for the software developer, and novel 'AI-friendly' software development incorporating developer interaction to foster engagement, trust, comfort and satisfaction. Addressing these research challenges, interactive approaches have emerged using a variety of metaheuristics including evolutionary computing [S1] and swarm intelligence [S8], and interactive evaluation mechanisms including, for example, weights [S9], scores [S24] and rankings [S25]. We speculate that this could reflect the emergence of a new subfield for SBSE, namely interactive SBSE (iSBSE). We are firstly motivated to conduct a systematic review to connect diverse studies that have hitherto largely been examined separately, thus being able to expose current approaches, open issues and future trends. Secondly, this review can provide some guidance to researchers on the development of interactive SBSE approaches, whose main components will be categorized here. We also seek to provide an understanding of the emerging subfield of iSBSE as a prerequisite and promoter of further industrial adoption of SBSE.

1.3 Approach and Contribution

To make connections among the diverse areas of knowledge in interactive SBSE and offer an up-to-date comprehensive picture of the state-of-the-art, this paper offers a systematic literature review (SLR) of interactive SBSE. Drawing upon established, evidence-based software engineering systematic literature methodology [23], we locate and examine 669 papers published in conference and journal papers related to interactive SBSE. We define appropriate inclusion and exclusion criteria to select a set of 26 primary sources. We then analyze the content of the primary sources, and with reference to existing surveys in the field [2], [3], formulate a classification scheme to provide a comprehensive overview of contemporary research for interactive SBSE. In addition to quantitative analysis, we review qualitative aspects to identify trends, gaps, open issues and, in our opinion, significant future trends. As a result of this systematic literature review, the comprehensive knowledge provided can enable researchers to make informed and effective decisions regarding interactive SBSE, as appropriate to their problem context.

This SLR of iSBSE is posed under the formulation of the following four research questions (RQ):

RQ1: In what ways has interactivity been adopted within search-based software engineering? To answer this RQ, the proposed classification scheme serves to reflect the main characteristics of current developments in iSBSE.

RQ2: Which findings about search techniques and interactive approaches along the complete development cycle can be extracted from the current state-of-the-art? This RQ aims at bringing the current state of the use of interactive approaches in SBSE to light. To this end, primary studies are analyzed regarding the addressed SE problem, the proposed interactive mechanism and the defined experimentation framework.

RQ3: To what extent do the detected gaps hamper human interaction in SBSE? After thoroughly analyzing findings from RQ2, some gaps in the application of interactive approaches in SBSE can be identified. Then we speculate possible causes, as well as provide suggestions for improvement.

RQ4: What are the emerging trends and how might they be addressed in the future? Given that the combination of search-based software engineering and interactive optimization is still an emerging topic, this paper outlines potential future lines of research.

1.4 Organization

The remainder of the paper is organized thus. In Section 2, some background to search-based software engineering, search techniques and human interaction is provided. Next, Section 3 provides an overview of the review methodology

used in this paper, including the classification scheme formulated for analysis of primary sources. In Section 4, we present findings of quantitative analysis before outlining the wider findings of the review in Section 5. In Section 6, we reveal the results of cross category data analysis to illustrate possible gaps and limitations of interactive SBSE, while speculating on possible causes. Then, in Section 7, we identify open issues and future trends before considering threats to validity in Section 8. Lastly, we conclude in Section 9.

2 Background

This section introduces the main concepts and terminology related to search-based software engineering, as well as a brief description of the most commonly applied search techniques in the field. Then, interactive optimization methods are presented.

2.1 Search-Based Software Engineering

The term search-based software engineering was coined by Harman and Jones in 2001 to encompass the application of search and optimization techniques to automate SE activities or support engineers during their resolution [2]. Search algorithms seek for optimal or near optimal solutions to a given problem, meaning that the first step towards the adoption of a SBSE approach is to reformulate the SE task as a search problem, i.e. the representation of the real-world problem so that the algorithm can generate and transform candidate solutions. In addition, the definition of a quantitative function, known as the fitness function, is required to let the algorithm discriminate between promising and poor solutions. This notion of quality is usually defined only in terms of software metrics, even though software engineers could make use of other more subjective mechanisms to assess quality, too.

Nowadays, SBSE covers all the phases of the software development process. In fact, there are a number of surveys and reviews specifically focused on each stage, including requirements management [24], software design [25], [26], software testing [27] and maintenance topics [28], [29], among others. For these specific SE problems, the selection of requirements to be implemented in the next iteration according to the stakeholders' interests and budgetary availability (the so-called next release problem, NRP), the early class analysis guided by software metrics, the automatic generation of test cases based on code coverage, or finding the optimal sequence of refactoring operations are some illustrative examples of SE tasks already solved using search techniques. The flourishing of these SLRs show an increasing interest in systematically analyzing the state of the field from diverse perspectives, though none of them address interactive optimization in SBSE.

The more complex the SE problem to be faced, the more sophisticated the optimization approach is required. Therefore, initial approaches adopted in early stages of SBSE were progressively enhanced to deal with multiple constraints, incorporate preferences, cope with uncertainty or allow the simultaneous optimization of two or more objectives [30]. Interestingly, Harman et al. [2] also suggest some areas of SBSE they consider overlooked and/or emerging. Among others, the potential of 'interactive optimization' is recognized, where issues related to possible fatigue and learning-effect bias still need to be thoroughly studied.

Continuing challenges make SBSE an increasingly active research field [31], which is also gaining the attention of industry [32]. Additionally, incorporating the software engineer's expertise to the process is essential to reach a good solution, which goes beyond the scope of software metrics [33]. Even for those cases where software metrics seem to be commonly accepted, the search process does not guarantee that the result of the search problem will result in representative solutions to the human, who may consider other additional, qualitative measures [34].

2.2 Search Techniques

Within the artificial intelligence field, search techniques are usually classified according to their ability to find the optimal solution and the exploitation of additional information when exploring the search space, i.e. the set of possible solutions [35]. Optimality of the final solution can be only guaranteed by exact methods, such as, for example, direct search of tree structures, dynamic programming, and integer linear programming. Therefore, to avoid having to explore all the candidate solutions, heuristic methods introduced the concept of problem-specific knowledge, defining rules to select or discard solutions depending on the current state of the process. Greedy search is a well-known example of a heuristic method.

Metaheuristic algorithms are iterative and non-exact methods, whose results could vary from one execution to another because of their stochastic nature. They are characterized by their efficiency and flexibility when solving complex optimization problems [36]. As opposed to heuristic procedures, metaheuristics are problem-independent techniques that define general and intelligent mechanisms to explore the search space, which are often inspired by biological processes. These algorithms are usually classified according to the number of candidate solutions they can handle at the same time [37]. Thus, a single-solution based metaheuristic starts from an initial solution and aims for its optimization in each iteration of the search process. On the other hand, a population-based metaheuristic deals with a set of solutions, combining their intrinsic information to reach a satisfactory solution. Within each category, a broad range of techniques can be found depending on how the search trajectory is determined or the nature-inspired mechanism that is being simulated by the algorithm.

Evolutionary algorithms (EAs) and Swarm Intelligence (SI) are two extensively used approaches of population-based metaheuristics. On the one hand, an EA starts with the random initialization of a population of solutions, called individuals. Then, an iterative process simulates the natural evolution of a species: selecting parents, generating offspring via crossover and mutation, and preserving the fittest individuals according to the fitness function. On the other hand, SI is based on emulating the collective behavior of entities working together in order to achieve a particular goal. For instance, a representative case is ant colony optimization (ACO), which takes inspiration from the foraging behavior of ants. Artificial ants look for the solution to the

optimization problem as if they build the shortest path to the food source. Decisions on the most promising paths are based on a pheromone matrix that is dynamically updated considering areas already explored by other previous ants and the quality of their corresponding solutions.

A complementary categorization of search techniques relies on the nature of the optimization problem and, more specifically, on the number of objectives to be simultaneously handled. Thus, single-objective optimization problems are defined in terms of a unique objective, whereas multi-objective optimization problems are characterized by the presence of 2 or 3 objectives, often in conflict. Those problems requiring the definition of 4 or more objectives are currently recognized as many-objective [38]. This property affects the way a search algorithm solves the optimization problem. On the one hand, single-objective algorithms are able to directly compare candidate solutions using the fitness function. On the other hand, multi-objective algorithms [39] look for trade-offs between objectives and, consequently, require new strategies to decide which solutions should be kept in each step. Similarly, many-objective algorithms [38] still need to consider these trade-offs, though the problem complexity demands specific mechanisms to deal with the high dimensionality of the objective space.

2.3 Interactive Optimization

As described in [3], interactive optimization methods enable the user's active participation in the search process. Also known as 'human-in-the-loop' approaches, they are founded on three main pillars: the need for more realistic optimization models (with respect to problem representation and fitness function), the scope for improvement of the search efficiency, and the complete satisfaction of the user's expectations. Notice that the real problem under study often needs to be oversimplified in order to formulate its computational representation. This can limit the optimization method by making it unable to capture the complete decision context beforehand, e.g. when considering imprecise objectives or temporary constraints. Besides, users can often become frustrated and lack confidence in results because of the existing gap between the solutions automatically generated by optimization techniques and their realistic expectations. User participation is specially relevant in dynamic scenarios, where the expert's knowledge could be considered as a complementary source for the exploratory capability of the search procedure. However, other factors inherent in being human (e.g. uncertainty and fatigue) can appear and need to be thoroughly considered [40].

In order to involve the human in the search process, the optimization method is required to provide intermediate outcomes to enable him/her to better understand the current search state, as well as how the process itself is being conducted. The role of human consists in returning some feedback that will be considered somehow thereafter in the iterative process [3]. In this context, the taxonomy proposed by Meignan et al. [3] provides a primary source to properly classify current interactive approaches in SBSE. More precisely, Meignan et al. establish five categories of interactive methods:

- *trial and error,* consisting of interactive adjustment of parameters;
- *interactive reoptimization*, aiming at redefining the formulation of the optimization problem;
- *interactive multiobjective optimization,* wherein the user interactively determines an appropriate trade-off between objectives;
- *interactive evolutionary computation* (IEC) [12], in which the human serves as fitness function; and
- *human-guided search,* a local search procedure directly incorporating transformations approved by the human.

When designing these interactive methods, there are a number of additional requirements that need to be considered, such as the point in which interaction happens (e.g. occasional or periodical participation), the specific task undertaken by the human (e.g. comparison, evaluation or correction of solutions) or the nature of the information gathered from such interaction, among others. Although the taxonomy proposed by Meignan et al. is comprehensive and covers many general interactive methods, not all the additional requirements are considered. Therefore, it is necessary to adapt and extend the taxonomy to address the specific requirements of interactive SBSE studies.

Finally, it is worth mentioning that SBSE can also be applied in combination with other types of approaches such as machine learning and interactive methods, which is known as interactive machine learning [41]. In short, machine learning (ML) techniques typically seek to predict classifications of new data by learning from past experience. With user involvement, a ML algorithm is able to integrate additional valuable information in order to build more precise and realistic models while gaining further knowledge.

3 Review Methodology

The review method has been built upon *best practice* in systematic literature review [23], [42]. Recent SLRs in the fields of empirical software engineering [43] and search-based software engineering [26], [44] have been taken as a reference, too. This section explains in detail the methodology used for conducting this review on iSBSE. To address the precisely raised research questions, a method is defined and followed in order to determine how the literature revision is performed. Such a method includes mentioning the literature search strategy, explicitly enumerating the inclusion and exclusion criteria for the papers found, and determining how the review data are collected from the selected works.

3.1 Literature Search Strategy

All the revised papers have been queried from a wide range of scientific literature sources, to prevent relevant studies remaining hidden:

- *Digital libraries:* ACM Library, IEEE Xplore, ScienceDirect, SpringerLink.
- *Citation databases:* ISI Web of Knowledge, Scopus.
- *Citation search engines:* DBLP, Google Scholar.

- *Specialized sources:* SBSE repository maintained by the CREST research group at UCL[1], as a reference for the SBSE community.

To build the search strings, the three authors firstly defined a list of terms embracing the variety of both application domains and search techniques to be covered. Synonyms and keywords were derived from this list and logical operators (AND, OR) used to properly link search terms. Beforehand, a pilot search was conducted on all the data sources to verify the adequacy and effectiveness of the resulting search strings. Then, some minor changes were applied to the queries to avoid a number of unwanted outcomes being returned. The refined search strings are shown in Table 1. Notice that search string #1 looks for SBSE publications in the broadest sense, whereas the search string #2 looks for those publications that do not explicitly contain the term SBSE but still apply search and optimization techniques to a SE task. The ACM Computing Classification System[2] and the IEEE Taxonomy[3] were used to define the list of tasks. When required, a search string was adapted to the specific query language used by a particular engine or database. Searches were conducted at first by the first author, and then double-checked by the rest of authors. String adjustments were agreed by all authors.

After the execution of the search queries, a total of 653 references were returned. Following best practices in conducting SLRs [23], [42], all authors performed independent manual searches to complement automatic results. More specifically, web profiles of relevant authors and their networks were consulted, and cross-references were checked following a *snowballing* procedure. As a result, 16 new references were added. Table 2 shows the number of publications found per data source, including the manual search. Then, an initial manual examination of titles and abstracts enabled publications not related to SBSE to be discarded. In this way, the list of candidate papers decreased to 67 publications (see Table 2). Finally, the papers to be included in the review, known as primary studies, are rigorously arrived at by applying the corresponding inclusion and exclusion criteria (see Section 3.2). A total number of 26 primary studies is obtained. During this search period monitoring and follow-up meetings were held to detect errors, disagreements or deviations from planned procedure.

3.2 Inclusion and Exclusion Criteria

From the list of candidate papers aforementioned, only those publications related to iSBSE could be considered as primary studies, i.e. the authors should propose an interactive search-based approach to solve a software engineering task. Such a limitation could be broken down into the following inclusion criteria:

1) The search model should explicitly incorporate a user, who is required to perform at least one interaction in order to solve a task during the search process, e.g. evaluation or selection of candidate solutions. Another valid approach is the user to participate at least once the search has finished but only when another iteration or algorithm is executed afterwards, so that the user's opinion could be integrated somehow in the overall process.
2) A paper should describe the method followed for the user's interaction. It should explain the role of the human in the search process, how his/her feedback is then integrated into the search process, or any other relevant aspect concerning the interactive action.
3) The search problem should be defined in terms of a decision space, evaluation objectives and the existing constraints, if any. The problem could be formulated as either single-objective or multi/many-objective, or both.
4) The search problem should be framed within one or several phases of the software development life cycle.
5) The search technique applied to address the software engineering problem is not restricted, but it should be a computational method.
6) The nature of the information provided by the user is not restricted, but it should cause an effect on the search.
7) Theoretical proposals are allowed.

In contrast, papers meeting any of the following exclusion criteria were not considered as a primary study:

1) The interactive approach does not address a software engineering optimization problem.
2) The user is only able to (re)configure the parameters of the search algorithm right before/after an independent run, or he/she is only able to make a decision at the end of the process, e.g. by selecting one single individual from the resulting solution set generated after completing a multi-objective optimization algorithm.
3) The research paper is written in a language other than English.
4) The full text of the manuscript is not accessible.
5) Either the publication process has not followed an accurate scientific peer-review process or there is no clear evidence of this point.

Finally, if multiple variants of the same research work are found, the conditions under which a paper is considered as a primary study are described next. They are applied if the same authors have published different papers for the same interactive approach, so only significant contributions are analyzed for the review. Nonetheless, notice that these constraints are also influencing the statistical study shown in Section 4:

- Considering a previous primary study, if the problem is the same but the technique is different and novel, then accept.
- Considering a previous primary study, if both the problem and the technique are the same but the authors have reported different findings from a new significantly different experimentation, then accept.
- If a journal paper is found as an extension of a previous conference paper and it satisfies the inclusion

1. http://crestweb.cs.ucl.ac.uk/resources/sbse_repository
2. https://www.acm.org/publications/class-2012
3. https://www.ieee.org/documents/taxonomy_v101.pdf

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSE.2018.2803055, IEEE Transactions on Software Engineering

TABLE 1
The two search strings defined

Search string #1	(interactivity OR interactive OR human-in-the-loop OR user-interaction OR user-centered OR user-centred) AND (search-based OR "search based") AND "software engineering"
Search string #2	(interactivity OR interactive OR human-in-the-loop OR user-interaction OR user-centered OR user-centred) AND software AND (requirements OR architecture OR design OR development OR testing OR debugging OR verification OR maintenance OR evolution) AND (search OR search-based OR optimisation OR optimization)

TABLE 2
Number of references per data source

Data source	Search results	Candidate papers	Primary studies
ACM Library	86	11	4
DBLP	20	15	5
IEEE Xplore	122	15	5
ISI-WoK	207	21	10
SBSE Repository	33	20	10
ScienceDirect	43	5	4
Scopus	218	32	13
SpringerLink	124	15	7
Manual search	16	15	6
Total unique papers	669	67	26

criteria, then reject the previous conference paper unless it provides with different but significantly distinct experimental outcomes.

3.3 Data Collection Process

After identification, primary studies were thoroughly analyzed to rigorously conduct the review process. With this aim, these papers are randomly distributed among the authors following some basic rules: (a) only the authors (3) of this review could participate in data collection process; (b) every primary study should be scrutinized by at least two reviewers; (c) a paper could not be reviewed by any of its authors; (d) there should be a balance among reviewers with respect to the number of papers, their category (i.e. conference, journal, etc.) and their SBSE topics that they study. Each reviewer should compile a data extraction form for each primary study. After completing the analysis, these forms are brought together to detect any disagreements between the reviewers. If so, the author who did not participate in the data extraction of the corresponding paper should also read the primary study and provide an opinion before a final and collective decision is reached.

All papers have been objectively reviewed under strict control and consistent conditions. No information has been inferred during the extraction process, and authors of the primary studies were not contacted. In the case of a potential conflict of interest when a primary study was authored by a reviewer, she/he did not participate in the data extraction and analysis. In addition, missing data are possible for some categories, and are reflected accordingly.

At this point, it is worth highlighting that data extraction discrepancies appeared for many of the primary studies, most of which were minor disagreements that could be resolved after a brief discussion between the reviewers involved in the data extraction process. In general, discrepancies appeared in all the categories. However, the disagreements appearing most frequently were focused on very specific aspects. For example, characterizing the role of the user, the interaction mechanism (type of solution shown and frequency of interaction), the influence of the user's opinion and the scope of the case studies were the most contentious discussion points. We observed that deciding the role of the user and the influence of his/her opinion were too often subject to the reader's interpretation. We also observed that the description of information such as the interaction mechanism was incomplete or imprecise in most primary studies. Furthermore, the lack of agreed benchmarks and standards within the SBSE field often made difficult to determine the scope of the cases studies. This emphasizes the importance of following a strict conceptual framework and agreed guidelines when defining the interactive proposal, given that misunderstanding can be easily reached otherwise.

3.4 Classification Scheme

In order to facilitate systematic data collection, as a response to RQ1, we developed a classification scheme drawing on best practice [23]. This scheme classifies iSBSE data relating to the following aspects: (i) *meta-information*, (ii) *problem formulation*, (iii) *search technique*, (iii) *interactive approach* and (v) *experimental framework*. For each of these aspects, we determined a number of data categories, and enumerated discrete values for each one. The resulting classification scheme does not only provide the guidelines for reviewers during the data extraction process, but also is intended to assist those researchers interested in iSBSE to accurately define their new approaches.

Firstly, meta-information related to the primary sources includes: (i) author(s) name(s); (ii) title of publication; (iii) type of publication (journal, conference or workshop); (iv) name of publication and publisher; (v) year of publication; and (vi) volume, issue and pages. For the sake of brevity, a brief introduction to the rest of categories follows below. Notice that a summary of the classification scheme can be found in the supplementary material, while a description is available on a website[4] accompanying this paper.

3.4.1 Classification of the Problem Formulation

The problem formulation refers to information describing the SE problem under study and its computational representation. Drawing on classifications used in previous surveys by Harman et al. [2] and Li et al. [38], and established software engineering texts [45], data are classified into 4 categories:

4. http://www.uco.es/grupos/kdis/sbse/isbse

- *Type of software engineering problem*, which refers to the phase of the software lifecycle where the problem is framed including values such as *requirements*, *analysis and design*, etc.
- *Development practice*, which corresponds to the SE methodology in which the problem is contextualized (*prescriptive model*, *specialized model* or *agile development*), if any.
- *Number of objectives* stated in the definition of the problem. Depending on this number, the problem is classified as *single-objective* (1 objective), *multi-objective* (from 2 to 3) or *many-objective* (more than 3).
- *Constraints*, which indicates whether the optimization problem comprises a formal definition of constraints (i.e. *constrained problem*). If there is an explicit mention to the lack of constraints, it will be defined as an *unconstrained problem*. Otherwise, N/A (not applicable/not available) is specified.

3.4.2 Classification of the Search Technique

Drawing on previous surveys of optimization heuristics [3], [37] and established texts relating machine learning to software engineering [46], search technique data are classified along 2 categories:

- Search *algorithm type*, which describes the resolution technique applied to the problem. The specific values correspond to techniques like *exact procedure*, *heuristic procedure*, *metaheuristic procedure* and *machine learning*, as introduced in Section 2.2.
- *Objective approach*, which specifies the number of objectives with which the algorithm deals, independently of the number of objectives defining the SE problem. The possible values are *single/multi/many-objective*, where single-objective could also include the use of an aggregation fitness function.

3.4.3 Classification of the Interactive Approach

The taxonomy proposed by Meignan et al. [3] provides a baseline classification for the interactive approach, which has been significantly extended to include additional information specifically adapted to the iSBSE field. We formulated 12 categories of data specific to iSBSE, as follows.

- The *interactive algorithm type* (adapted from [3]) differentiates among four interactive optimization methods: (i) *interactive reoptimization*, where the problem definition is refined; (ii) *preference-based interactivity*, where the goal is to incorporate users preferences during the search; (iii) *human-based evaluation*, wherein the user totally or partially replaces the fitness function; and (iv) *human-guided search*, where user actions directly impact candidate solutions. These methods can appear in combination.
- Two non-exclusive *purposes of user interaction* are defined to describe the focus of such interaction [3]. On the one hand, an interaction is *problem-oriented* when the user contributes to (re)define the optimization problem. On the other hand, a *search-oriented* interaction is focused on actions conducted during the search procedure in order to improve its efficiency.
- The *role of the user* in the process depends on the specific selected purpose of interaction [3]. Hence, if it is a problem-oriented interaction, the user could be an *adjuster* of constraints and/or objectives, or and *enricher* who adds or removes them. For a search-oriented interaction, there are three possible roles: (i) *assistant*, which refers to the selection or modification of solutions; (ii) *guide*, which means that the user controls the search process; or (iii) *tuner*, relating to adjusting search parameters.
- The *type of user task* describes the action(s) performed by the user over the solution(s). It includes *evaluation* of some aspects of the solution quality, the *selection* of promising solutions, the *comparison* between two or more solutions, or even the manual *modification* of the solution, including freezing it. More than one task could be performed if requested.
- The *evaluation mechanism* should be identified if any sort of evaluation is involved in the interaction. The possibilities consist of (i) providing a *fitness* value, (ii) assigning *weights*, (iii) offering a discrete *score*, (iv) giving *rankings*, and promoting or demoting with a *reward or penalization*.
- The *adjustment of interaction time* describes how often interaction occurs. If interruptions are *fixed* a priori, an interaction could occur *every iteration*, *every N iterations* or *between two runs* of an algorithm. Otherwise, if the interaction is *dynamic*, then it could be either based on the course of the search (*adaptive*) or requested by the user (*on demand*).
- The *number of solutions* shown to the user per interaction can be set to only *one*, a *pair*, *N* solutions or *all* of them, i.e. the whole population.
- The *level of detail* indicates whether the information shown to the user about a given solution for a specific interaction represents a *complete* or a *partial* solution.
- The *selection strategy* for solutions to show to the user can rely on the algorithm (*fixed*) or the user (*free*). In the first case, the specific criterion is determined by selecting the *best*, *random* or a *specific* solution. It is also possible to show *all* of them.
- *Feedback integration* refers to the mechanism that manages the users opinion [3]. It is defined as *model-free* when the preference information is directly incorporated by the algorithm. In contrast, *model-based* approaches include a learning process instead.
- The *preference information lifetime* describes the temporal scope of users feedback [3]. Information lifetime can be *step based* (used between two sequential interactions, (ii) *short-term* (valid for a single run or (iii) *long-term* (reused across different executions).
- *Information validity* indicates how the information is integrated and kept during the search. Three values are possible: (i) *permanent*; (ii) *flexible*, if the user can modify feedback; and (iii) *unrestricted*, if the user can revoke feedback.

3.4.4 Classification of Experimental Frameworks

Categories regarding the experimental framework are extracted after a comprehensive examination of primary stud-

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSE.2018.2803055, IEEE Transactions on Software Engineering

ies and for the statistical analysis, the guide by Arcuri and Briand [47] was considered. We formulated 8 categories of data to classify the experimental frameworks used in the primary studies, as follows.

- *Type of study*, which determines the scope of the experimental study presented in the paper. Four values have been identified: (i) *theoretical proposal*, (ii) *sample execution*, (iii) *simulated interaction* and (iv) *empirical investigation*.
- For empirical studies, the *number of participants* should be specified using prefixed intervals.
- For empirical studies, the *participant position* should be determined to develop a complete profile of the users. Accepted values include: (i) *engineer*, (ii) *PhD academia staff*, (iii) *postgraduate student*, (iv) *undergraduate student*, and (v) any *other*.
- For empirical studies, total work experience (*expertise*) of the users should be also gathered in terms of time intervals varying between *less than 5 years* and *more than 20 years*.
- Experimental *evaluation criteria* should be selected to indicate how results are analyzed. The three criteria found in iSBSE studies are *measures* (e.g. fitness values), examples of *solutions* and responses to *questionnaires*.
- *Case studies* reported in experiments can be artificially created (*synthetic*), small but real problems known to participants (*controlled environment*) or an *industrial case*.
- If *evidence* of additional materials is found, the available information should be accessed, compiled and classified by their respective types (*source code*, *problem instances*, *raw results*, *solutions*, *questionnaires*, *transcripts*, *additional statistics* or *other*).
- Studies including *statistical tests* may contain one or more types of tests (*pairwise comparison*, *multiple comparison*, *effect size measurement*).

4 Quantitative Analysis

As an initial result of the analysis conducted to respond RQ2, this section provides quantitative information about iSBSE studies and their respective authors in order to provide an overview of the state of the field.

4.1 Information on Sources

After strictly applying the method described in Section 3.1, 26 primary studies have been obtained (a separate list of references is provided at the end of the paper). It should be noted that the 8 candidate papers shown in Table 3 also satisfied the inclusion criteria, but they were considered as variants of primary studies according to the review protocol (see Section 3.2). Although excluded from the review analysis, they have been included within this quantitative analysis to reduce the bias towards certain types of publication and authors.

Fig. 1 depicts the bar chart displaying the cumulative number of publications per year. As can be observed, the first two contributions within the field were published in 1999 by Monmarché et al., though no more publications

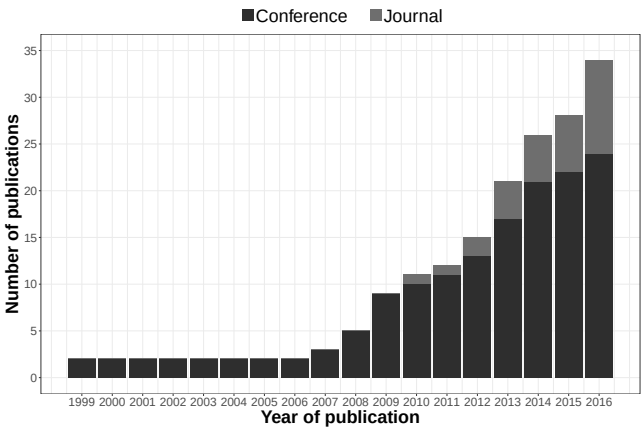


Fig. 1. Cumulative number of publications per year and type

appeared until the mid 2000s. In the earlier years, the number of papers showed a small but steady increase. However, after 2013, iSBSE attracted increasing attention. This is evidenced by how the different types of publication have evolved. Fig. 1 reveals that the majority of publications (71%) are contributions to international conferences. Nevertheless, 29% of the total of publications refers to journal papers. It is worth noticing that 8 out the 10 journal papers were published after 2013.

Regarding the journals where papers are published and, consequently, their audience, submissions are equally distributed, only *Applied Soft Computing* and *Information and Software Technology* appearing twice (see List of primary studies before References). Most of these articles are ranked top according to the *Journal Citation Reports*, covering a broad spectrum of areas: Software Engineering (3), Artificial Intelligence (2), Multidisciplinary Sciences (2) and Cybernetics (1). This seems to be clearly motivated by the multidisciplinary nature of SBSE. As for conference papers, a similar distribution between the SE and AI communities is found. However, it is worth mentioning a more frequent appearance of the *Symposium on Search-Based Software Engineering*, a specialized event where 3 papers were presented and extended afterwards as journal papers [S1], [S4], [S18].

4.2 Information on Authors

A total of 66 authors based in 13 different countries have been identified from the list of publications under study. Table 4 shows the most frequent authors in the area, including their number of co-authored publications. They contribute to 76% of primary studies and their variants. With respect to their country, Table 5 shows the total number of papers published in iSBSE per country. Authors based in Italy, Sweden, the United Kingdom (UK) and the United States of America (USA) are the most frequently published in iSBSE. With the aim of providing a baseline for SE publications, the affiliation countries and level of cooperation have been also analyzed from a renowned broad community like ICSE (International Conference on Software Engineering). Between 2007 and 2016, 845 ICSE technical papers were authored by researchers based in 41 different countries, USA, Canada and Germany being the most frequent. Both

TABLE 3
List of variants of primary studies sorted by publication year and first author's surname

Reference	Authors	Title	Journal/Conference	Year
[48]	A.A. Araújo and M.H. Paixão	Machine Learning for User Modeling in an Interactive Genetic Algorithm for the Next Release Problem	Symposium on Search Based Software Engineering	2014
[49]	B. Marculescu et al.	Practitioner-Oriented Visualisation in an Interactive Search-Based Software Test Creation Tool	Asia-Pacific Software Engineering Conference	2013
[50]	B. Marculescu et al.	A Concept for an Interactive Search-Based Software Testing System	Symposium Search Based Software Engineering	2012
[51]	P. Tonella et al.	Using Interactive GA for Requirements Prioritization	Symposium on Search-Based Software Engineering	2010
[52]	L. Troiano et al.	Interactive Genetic Algorithm for choosing suitable colors in User Interface	Learning and Intelligent Optimization Conference	2009
[53]	C. Simons and I.C. Parmee	Agent-based Support for Interactive Search in Conceptual Software Engineering Design	Genetic and Evolutionary Computation Conference	2008
[54]	C. Simons and I.C. Parmee	User-centered, Evolutionary Search in Conceptual Software Design	IEEE Congress on Evolutionary Computation	2008
[55]	N. Monmarché et al.	On Generating HTML Style Sheets with and Interactive Genetic Algorithm Based on Gene Frequencies	European Conference on Artificial Evolution	1999

TABLE 4
List of authors with three or more publications

Author	No. Publications
Allysson Araújo	4
Cosimo Birtolo	3
Altino Dantas	3
Robert Feldt	6
Marouane Kessentini	3
Bogdan Marculescu	5
Francis Palma	3
Ian Parmee	5
Christopher Simons	6
Jerffeson de Souza	3
Angelo Susi	3
Paolo Tonella	3
Richard Torkar	5
Luigi Troiano	3

TABLE 5
List of countries

Country	No. Publ.	Publication year
Brazil	4	2014-2016
Canada	1	2013
China	2	2013, 2016
Egypt	1	2014
Finland	1	2013
France	2	1999
Ireland	1	2014
Italy	7	2009-2013, 2016
Singapore	1	2013
Sweden	6	2009, 2012, 2013, 2015, 2016
Tunisia	1	2014
United Kingdom	7	2008-2010, 2012, 2014, 2016
United States of America	7	2007, 2013, 2014, 2016

the number of papers and countries show that iSBSE is still a relatively small area, but somehow more widely distributed by country. For instance, around 50% of ICSE papers have one or more authors affiliated to a USA institution, whereas authors based in USA appear in 21% of iSBSE papers. With a similar percentage in iSBSE, UK and Italy obtain the 5th and 6th positions for ICSE publications, respectively. The level of cooperation between different countries is slightly lower than that of the ICSE community, notably because iSBSE is still in its earlier stages. In this sense, 53% of the considered publications are co-authored by researchers working for different institutions (51% in ICSE), 39% of which refer to collaborations between authors located in different countries (55% in ICSE).

5 Findings of the Review Process

Responding to RQ2, this section provides a detailed review of the primary works under study. Following the high-level structure specified in the classification scheme described in Section 3.4, the data extraction process has produced insightful results regarding the problem formulations addressed in iSBSE, the search techniques applied, the mechanisms for human interaction, and how outcomes are empirically validated.

5.1 Problem Formulation

As shown in Table 6, iSBSE has been proposed to address a significant number of project phases. Tasks related to the analysis and design, where creativity and experience play a key role, have emerged as problem fields where human interactivity has been applied to achieve better search performance. In these cases, there is a variety of design artifacts to be optimized, including object-oriented specifications [S8], [S10], [S11], [S13], software architectures [S9], software product lines [S7] and graphic user interfaces [S6], [S12], [S14]. In contrast, other problem areas like testing and verification or coding, incorporate the software engineer's opinion in the search process to a limited extent. Notice that interactive search-based refactoring is classified under the area 'distribution and maintenance', wherein artifacts include both design models and source code. Others, such as those related to project management, are as yet unexplored. Indeed, the lack of research studies into interactive search-based project management is perhaps surprising, given that others have previously explored interactive search-based approaches for project management more generally.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSE.2018.2803055, IEEE Transactions on Software Engineering

TABLE 6
Problem formulation

Category	Percentage	Papers
Type of software engineering problem		
Requirements	19%	[S1], [S2], [S3], [S4], [S5]
Analysis and design	38%	[S6], [S7], [S8], [S9], [S10], [S11], [S12], [S13], [S14], [S15]
Code implementation	4%	[S16]
Testing and verification	12%	[S17], [S18], [S19]
Distribution and maintenance	27%	[S20], [S21], [S22], [S23], [S24], [S25], [S26]
Project management	0%	
Development practice		
Prescriptive process model	12%	[S1], [S2], [S3]
Specialized process model	4%	[S24]
Agile development	0%	
Not specified	85%	[S4], [S5], [S6], [S7], [S8], [S9], [S10], [S11], [S12], [S13], [S14], [S15], [S16], [S17], [S18], [S19], [S20], [S21], [S22], [S23], [S25], [S26]
Number of objectives		
Single-objective	27%	[S1], [S2], [S3], [S4], [S5], [S15], [S26]
Multi-objective	38%	[S6], [S8], [S10], [S11], [S12], [S13], [S14], [S19], [S20], [S23]
Many-objective	23%	[S7], [S9], [S17], [S18], [S24], [S26]
Not specified	15%	[S16], [S21], [S22], [S25]
Constraints		
Constrained	38%	[S1], [S2], [S3], [S4], [S5], [S8], [S10], [S11], [S13], [S22]
Unconstrained	27%	[S7], [S12], [S15], [S19], [S23], [S24], [S26]
Not specified	35%	[S6], [S9], [S14], [S16], [S17], [S18], [S20], [S21], [S25]

Only 15% of sources specify the *development practice* for interactive search (see Table 6). In such a case, an iterative and incremental process model is indicated for the majority of sources, all of them dealing with the next release problem [S1], [S2], [S3], while one source reports a specialized process for model driven engineering [S24]. Agile development is not specifically reported as a software engineering process in any source.

Regarding the *number of objectives* reported for each problem context, the majority (i.e. 62%) of sources describe a multi/many objective software engineering problem, compared to (27%) that report problems formulated as single objective. This finding is in line with the general trend of SBSE, where the adoption of multi-objective problem formulations is increasing in order to better reflect the multiple decision factors involved in any SE problem [30]. Besides, some studies i.e. [S16], [S21], [S22], [S25] are not sufficiently informative about the number of objectives, which is noteworthy in a field like SBSE. With respect to the applicability of *constraints* in the problem formulation, fewer than half of the sources (38%) can be properly described as constrained. Notice that the constrained problems mainly refer to the NRP [S1], [S2], [S3], [S4], [S5], in which budgetary restrictions need to be considered, and design tasks [S8], [S10], [S11], [S13], which usually define specific rules to derive valid specifications. The remainder may either be described as unconstrained, or the application of constraints is not mentioned in the source.

5.2 Search Technique

Table 7 classifies the primary studies according to the search methods they apply and objective approach (see Section 3.4). In terms of the *algorithm type* used as a search technique, it can be observed that metaheuristics are the most frequently applied (88%). Within this group, evolutionary computation is used by a large majority of primary sources (81%), though different types of evolutionary algorithms are considered, including genetic algorithms (57%), multi-objective evolutionary algorithms (24%), differential evolution (14%) and genetic programming (5%). This may largely be due to a historical bias to evolutionary computing among metaheuristics carrying through to SBSE [2]. Nevertheless, two studies based on the application of swarm intelligence follow the ACO paradigm. In the minority is one paper that uses exact search, and two that use machine learning. None of the primary studies rely on heuristics to conduct the search.

With respect to the *objective approach* of the search techniques described, the majority of sources (73%) describe a single-objective search approach, many of them using an aggregation of several metrics to compute fitness. The remainder describe multi/many-objective search, with the exception of one paper [S15] not defining a fitness function and ML approaches. As can be seen from Table 7, two primary studies [S11], [S26] explore the corresponding SE problem from two different perspectives. Even though most of the metaheuristic paradigms have been adapted to deal with multiple objectives, only evolutionary techniques have been applied in iSBSE hitherto, due presumably to their broad popularity and maturity.

5.3 Interactive Approach

Firstly, we examined the *interactive algorithm type* used in the interactive approach with the software engineer. As shown in Table 8, at 42%, interactive reoptimization and preference-based interactivity are the most frequently occurring type

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSE.2018.2803055, IEEE Transactions on Software Engineering

TABLE 7
Search techniques currently used in iSBSE

	Single-objective	Multi-objective	Many-objective	Not applicable
Exact	[S5]			
Metaheuristic				
Single-solution based	[S20]			
Evolutionary computation	[S1], [S3], [S6], [S17], [S18], [S19], [S20], [S21], [S22], [S24] [S4], [S9], [S11], [S12], [S14], [S26]	[S10], [S23], [S26] [S11], [S13]	[S7]	[S15]
Swarm intelligence	[S2], [S8]			
Machine learning				
Supervised				[S16]
Unsupervised				[S25]

TABLE 8
Interactive approaches and their application in iSBSE

Interactive algorithm type	Description	Percentage	Papers
Interactive reoptimization	Feedback is used to modify some elements of the problem formulation (adding constraints or objectives, adjusting weights, etc.) instead of the solution itself.	42%	[S2], [S3], [S4], [S5], [S7], [S9], [S10], [S17], [S18], [S19], [S20]
Preference-based interactivity	Preference information is used to guide the search towards some zones of search/objective space, e.g. selecting between two candidate solutions.	42%	[S7], [S11], [S13], [S14], [S15], [S16], [S20], [S21], [S23], [S25], [S26]
Human-based evaluation	Human subjective judgment is obtained by means of different evaluation mechanisms (e.g. scores, rankings, etc.)	27%	[S1], [S4], [S6], [S8], [S12], [S22], [S24]
Human-guided search	Actions performed by the human that have a direct impact on the solutions, such as freezing parts of the encoding.	8%	[S9], [S15]

TABLE 9
Purpose of interactivity and role of the user

Category	Percentage	Papers
Problem-oriented interaction	69%	
Adjuster	35%	[S2], [S8], [S9], [S10], [S14], [S17], [S18], [S19], [S21]
Enricher	35%	[S1], [S3], [S4], [S5], [S6], [S7], [S20], [S24], [S26]
Search-oriented interaction	50%	
Assistant	12%	[S8], [S15], [S16]
Guide	23%	[S7], [S11], [S12], [S13], [S22], [S23]
Tuner	19%	[S1], [S2], [S9], [S15], [S25]

of algorithm used. Human-guided evaluation is described in 27%, while human-guided search is employed in two papers. Notice that five primary studies [S4], [S7], [S9], [S15], [S20] combine two different interactive approaches, which seems to indicate that the authors prefer to focus the interaction on a specific activity or part of the search process. Interestingly, iSBSE studies do not usually delegate the complete evaluation of solutions as originally proposed by IEC, allowing users to influence the search in more flexible ways, such as adding restrictions or selecting solutions.

Secondly, for the *purpose of user interaction* (see Table 9), a greater proportion (50%) of approaches are problem-oriented compared to search-oriented (31%). As the former type of interaction is concerned with the application domain, gathering knowledge from the expert seems to be a primary goal when adopting an interactive method. Nevertheless, there are also some primary studies (19%) combining both approaches, which allows the decision maker to perform different actions. In fact, the *role of the user* in interaction shows some variety. A more in-depth view reveals that objectives are often adjusted by means of weights [S8], [S19]. However, the addition of constraints [S3], [S4] and providing a subjective evaluation of solutions to complement their fitness value [S1], [S24] are frequent actions when the user acts as enricher. Among search-oriented interactions, there are diverse approaches. When the user is an assistant, the interaction consists in finding promising solutions by marking some of them [S8] or labeling examples [S16]. Guiding the algorithm towards specific regions of the search space often appears in multi-objective approaches [S7], [S23], while tuners are allowed to dynamically change parameters such as the probabilities associated with genetic operators [S9], [S15], [S25].

As shown in Table 10, more than a half of the sources describe evaluation of solutions as the *type of user task*. Selection, comparison and modification of solutions are described in 31%, 12% and 23% of sources respectively. Such a variety might reflect the emerging nature of this research field, wherein no single approach to user interaction with search is yet predominant. In fact, even though manually fixing solutions could be viewed as a natural action to be performed, modifying solutions mostly appears in combination with other tasks like evaluation and comparison.

When the provided feedback directly or indirectly influences the evaluation phase, the user applies a particular *evaluation mechanism* to achieve this. Table 10 shows that

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSE.2018.2803055, IEEE Transactions on Software Engineering

TABLE 10
Tasks performed during interaction and evaluation mechanisms

Category	Percentage	Papers
Type of task		
Evaluation	58%	[S1], [S2], [S3], [S6], [S8], [S9], [S10], [S12], [S17], [S18], [S19], [S21], [S22], [S23], [S24]
Selection	31%	[S7], [S8], [S11], [S13], [S14], [S15], [S16], [S25]
Comparison	12%	[S4], [S5], [S26]
Modification	23%	[S8], [S9], [S15], [S20], [S23], [S26]
Evaluation mechanism		
Fitness value	4%	[S22]
Weights	15%	[S9], [S17], [S18], [S19]
Scores	31%	[S1], [S6], [S8], [S10], [S11], [S12], [S13], [S24]
Rankings	8%	[S4], [S25]
Reward/penalization	42%	[S2], [S3], [S5], [S13], [S14], [S15], [S16], [S20], [S21], [S23], [S26]
Not applicable	4%	[S7]

reward/penalization is most frequently occurring at 42% of sources, followed by scores (31%), and weights (15%). Ranking is described in two sources while providing a fitness value is described in one.

Another key aspect of the interactive approach is the *adjustment of interaction time*. Regarding the number of iterations/generations of search between user interaction, the great majority of sources (92%) report a fixed number of iterations (see Table 11). Conversely, only two studies apply an adaptive approach, where the time of interaction depends on either how fitness values are progressing [S8] or when a tie is found [S4].

The primary studies provide further detail about the way the user interacts the algorithm, and how solutions are shown to the decision maker. Regarding the *number of solutions* presented at an interactive event, Table 11 shows that 27% sources describe the presentation of one solution to the user for evaluation, the remainder describe multiple solutions presented (with the exception of one paper that does not provide any information in this regard). Notice that one significant case of multiple solutions is when the user is requested to pick a pair of solutions from the candidate set in order to compare them [S4], [S5]. The *level of detail* of the presentation to the user is mainly complete detail (at 77%), and only four studies [S5], [S16], [S21], [S26] focus the interaction on specific parts of the solutions. A combination of both approaches can be also found in two studies [S17], [S18], where the user is able to get an overview of the candidate solutions in terms of their objective values before entering to inspect the most interesting ones in more detail.

In 88% of sources, the *selection strategy* to pick solutions was controlled by the algorithm. It appears that interaction with the user is mostly viewed as a fixed mechanism to reinforce some aspects of the automatic search, instead of an opportunity to create a more flexible, collaborative, user-oriented process. Selecting best, specific or all solutions are equally considered (27%). However, random selection of solutions is not described. Some studies propose returning specific solutions by selecting those with the same fitness value so that the user is requested to determine a winner [S4], [S5]. In [S12] a clustering technique is applied to perform the selection. Notice that only one primary source describes the user providing a qualitative fitness value. We suggest that this, combined with a variety of level of solution detail presented, may be an attempt to capture implicit preference information [14] from the user.

Analysis of primary sources, compiled in Table 12, shows that the *feedback integration* within search directly affected the solution models in 19% of sources, compared to the great majority (i.e. 81%) of sources that are reportedly model-free. Although the latter approaches use ML techniques with different purposes, most of them are focused on modeling the subjective evaluation. For instance, two studies [S8], [S10] propose adjusting the weights associated with the metrics comprising the fitness function by applying a regression model of the user's ratings. Two other studies [S1], [S22] replace user's feedback by using neural networks to build their learning models. In [S1], a regression method (least mean square) is also considered for comparison.

With respect to the *preference information lifetime* of user feedback during search, it is observed that sources describe either a step-based (50%) or short-term (54%) lifetime. However, long-term lifetimes are not described, that is, information is not reused across different runs of the algorithm to solve similar problems. In addition, with respect to the *information validity* of the user feedback, permanent and flexible account for 54% and 42% of sources respectively. Notice that these approaches somehow restrict the information flow between the algorithm and the user, who is likely to be interested in ensuring that his/her opinion actually influences the search or changes it in light of new results. Only four sources [S8], [S15], [S16], [S25] allow the user to revoke the decision made during a previous interaction.

5.4 Experimental Framework

Regarding the *type of study* used in experiments, Table 13 indicates that empirical investigations are described in over half of the primary sources (65%). Simulated interaction is reported in 35% of sources, while sample execution and theoretical proposals account for 4 sources and one source respectively. Table 14 shows the number of participants and their specialization for those primary studies reporting some kind of empirical investigation. With respect to the *number*

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSE.2018.2803055, IEEE Transactions on Software Engineering

TABLE 11
Selection of solutions and adjustment of interaction time

Adjustment of interaction time		
Fixed	92%	
Every iteration	19%	[S1], [S5], [S10], [S15], [S22]
Every n iterations	35%	[S2], [S6], [S12], [S14], [S17], [S18], [S19], [S24], [S26]
Between two runs	38%	[S3], [S7], [S9], [S11], [S13], [S16], [S20], [S21], [S23], [S25]
Dynamic	8%	
Adaptive	8%	[S4], [S8]
On demand	0%	
Category	**Percentage**	**Papers**
Number of solutions		
One solution	27%	[S2], [S3], [S8], [S9], [S10], [S20], [S26]
Pair of solutions	8%	[S4], [S5]
N solutions	38%	[S1], [S6], [S11], [S12], [S13], [S14], [S15], [S19], [S22], [S24]
All solutions	23%	[S7], [S17], [S18], [S21], [S23], [S25]
Not specified	4%	[S16]
Level of detail		
Complete solution	77%	[S1], [S2], [S3], [S4], [S6], [S7], [S8], [S9], [S10], [S11], [S12], [S13], [S14], [S15], [S19], [S20], [S22], [S23], [S24], [S25]
Partial solution	15%	[S5], [S16], [S21], [S26]
Both	8%	[S17], [S18]
Selection strategy		
Fixed	88%	
Best solution(s)	27%	[S2], [S3], [S9], [S19], [S20], [S24], [S26]
Random solution(s)	0%	
Specific solution(s)	27%	[S4], [S5], [S8], [S10], [S11], [S12], [S21]
All solutions	27%	[S1], [S7], [S15], [S17], [S18], [S23], [S25]
Not specified	8%	[S14], [S22]
Free	8%	[S13], [S16]
Not specified	4%	[S6]

of participants in experiments, only 8% of sources included greater than 20 participants. In fact, acquiring a large number of diverse participants appears to be a major challenge in this area. From the studies including some empirical investigations, 88% of sources either partially or completely describe the profession and expertise of the participants.

Analysis of the *participants' position* also reveals that 53% of the empirical investigations are conducted with participants who came from an academic background, either as staff (12%) or students (35%). 35% of the empirical studies describe their participants as being software engineering practitioners in an industrial setting. Furthermore, in another four studies [S9], [S10], [S20], [S23], we found other user profiles, although all participants perform the same experimental task. In other cases such as [S17], [S20], [S25], participants are separated into experimental and control groups in order to compare the interactive experience against manually resolving the corresponding task. It is noteworthy that empirical investigations are occasionally conducted in combination with other methods like simulated interactions [S1], [S2], [S18] or sample executions [S9], which provide an efficient comparison framework. In commenting on these findings, we note that the majority of experiments report empirical investigations, providing a necessary level of robustness and rigor associated with experiments involving human behavior.

The *evaluation criteria* determine which mechanism is used to analyze the experimental outcomes. In interactive search experiments (see Table 13), the majority of the analyzed sources (81%) report the use of measures, i.e. fitness values or specific assessment metrics. With respect to the scope of the software artifacts deployed as *case studies* in experiments, 58% of sources report controlled environments. On the other hand, industrial case studies are reported in 38% of sources, and four sources describe the use of synthetic case studies. We note that, while acknowledging the logistical challenges involved, it is clear that from the point of view of iSBSE as a requisite and precursor to the adoption of industrial SBSE tools, experiments involving industrial case studies are to be preferred over controlled environments and synthetic simulations.

It is remarkable that only some studies are accompanied with *evidence* in the form of additional experimental material. Table 13 reflects the nature of the reachable materials, which were found in 27% of sources. Finally, we find that 58% of studies employ *statistical tests* to validate experimental results. Pairwise and/or multiple comparison techniques are reported in 15 sources, while estimation of effect size is conducted in 3 (12%). Table 15 shows the details of the specific statistical tests applied in each case study. As can be observed, the Wilcoxon Mann-Whitney test is preferred for pairwise comparisons. There is less agreement for multiple

TABLE 12
Feedback integration and information lifetime

Category	Percentage	Papers
Feedback integration		
Model-free	81%	[S2], [S3], [S4], [S5], [S6], [S7], [S9], [S11], [S12], [S13], [S14], [S15], [S17], [S18], [S19], [S20], [S21], [S23], [S24], [S25], [S26]
Model-based	19%	[S1], [S8], [S10], [S16], [S22]
Preference information lifetime		
Step-based	50%	[S4], [S6], [S7], [S12], [S13], [S14], [S15], [S17], [S18], [S19], [S21], [S23], [S24]
Short-term	54%	[S1], [S2], [S3], [S5], [S8], [S9], [S10], [S11], [S15], [S16], [S20], [S22], [S25], [S26]
Long-term	0%	
Information validity		
Permanent	54%	[S1], [S4], [S5], [S6], [S9], [S11], [S13], [S14], [S20], [S21], [S22], [S23], [S24], [S26]
Flexible	42%	[S2], [S3], [S7], [S8], [S9], [S10], [S12], [S15], [S17], [S18], [S19]
Unrestricted	15%	[S8], [S15], [S16], [S25]

comparison or effect size measurement, and their use is not that usual either. In commenting on these findings, we would encourage authors to make experimental materials available for readers. Furthermore, in recognition of the stochastic nature of metaheuristic search and the variability of human behavior, we would also encourage authors to always make use of appropriate statistical analysis.

6 Cross Category Data Analysis

As a response to RQ3, in this section we review the extracted data across categories of the classification scheme as a way to identify potential gaps and limitations of iSBSE, while speculating about possible causes. We firstly compare and contrast extracted data relating to the software engineering *problem formulation* with data in the various categories of *search technique* and *interactive approach* used in the primary sources. Secondly, we contrast the data regarding *search techniques* in the primary sources with the data in various categories of *interactive approach*. To illustrate possible associative relationships, bubble charts revealing the frequency of occurrence of data from various categories are shown. Note that statistical analysis was considered as a means to determine the presence of correlation. However, due to sparsity and a lack of independence of the categorical data, this proved inappropriate.

6.1 Problem Formulation and Approaches

Fig. 2(a) reveals the frequencies of search techniques used in various problem formulations. As can be seen, evolutionary computation is widely adopted in almost all SE phases, but is frequently used in analysis and design (9 studies) and maintenance (6 studies) problems. It is worth noting that other metaheuristics like swarm intelligence [S8] and hill climbing [S20] have been also applied very recently for the first time in the area. For instance, first results on the application of swarm intelligence for requirement elicitation were published in 2016 [S2].

Fig. 2(b) shows how a great variety of types of interactive algorithms have been already employed to address analysis and design problems. We observe that iSBSE was first used in design tasks. As a result, the landscape of approaches in this area is more extensive, including the only two primary studies on human-guided search [S9], [S15]. An interesting finding here is that, even though the automatic evaluation of solutions is a well-known challenge in SBSE, it remains a difficult task for the software engineer as 'human-in-the-loop', and so indirect mechanisms to evaluate candidate solutions are preferred in iSBSE. This is reflected by the fact that PI (preference-based interactivity) and IR (interactive optimization) are more frequently used than HE (human-based evaluation) in all SE areas. As for PI, it is the most frequent interactive approach for the design and maintenance categories of problem formulation, meaning that researchers find it appropriate to leverage the designer's abilities for recognizing promising solutions. It is noteworthy that this approach is not used for requirements and testing problems, wherein human preferences often abound.

Fig. 2(c) reveals that EV (evaluation) is frequently used across a range of problem formulations (with the exception of source code development). This might reflect a relative ease of implementation by simply presenting candidate solution(s) in a visualization for either direct or indirect evaluation, when compared with other interactive tasks. An interest in gathering information to reinforce the problem formulation predominates. It is also noticeable that CO (comparison) is not found in relation to analysis and design. Apparently, this task might be well suited to design problems, though the implementation difficulties caused by visually presenting multiple complete solutions might represent the major limitation at this point.

Fig. 2(d) illustrates the frequencies of interactive evaluation mechanisms used across various problem formulations. We can observe here that evaluation based on RA (rankings) is the least frequently occurring, and is confined to requirements and maintenance problems. It is interesting to speculate that this may in some part be due to the relative difficulty of implementing a RA mechanism as part of the visualization of multiple candidate solutions for an interactive user experience. To overcome such a limitation, the amount of information to be shown needs to be reduced and somehow limited to a subset of solutions. For example, the sorting procedure applied to requirements prioritization [S4] only displays those solutions showing a conflict in the priority order within a pair of requirements. In addition, the information is limited by setting a maximum number of

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSE.2018.2803055, IEEE Transactions on Software Engineering

TABLE 13
Characteristics of the experiments conducted in iSBSE

Category	Percentage	Papers
Type of study		
Theoretical proposal	4%	[S7]
Sample execution	15%	[S9], [S11], [S15], [S20]
Simulated interaction	35%	[S1], [S2], [S3], [S4], [S5], [S12], [S18], [S19], [S26]
Empirical investigation	65%	[S1], [S2], [S6], [S8], [S9], [S10], [S13], [S14], [S16], [S17], [S18], [S20], [S21], [S22], [S24], [S25]
Evaluation criteria		
Measures	81%	[S1], [S2], [S3], [S4], [S5], [S6], [S8], [S10], [S11], [S12], [S13], [S14], [S16], [S17], [S18], [S19], [S22], [S23], [S24], [S25], [S26]
Solutions	35%	[S2], [S9], [S11], [S14], [S15], [S16], [S20], [S21], [S23]
Questionnaires	35%	[S1], [S6], [S8], [S10], [S16], [S17], [S18], [S23], [S25]
Case studies		
Synthetic	15%	[S1], [S2], [S9], [S15]
Controlled environment	58%	[S2], [S3], [S8], [S10], [S11], [S14], [S16], [S17], [S20], [S21], [S22], [S23], [S24], [S25], [S26]
Industrial case	38%	[S1], [S4], [S5], [S6], [S13], [S18], [S19], [S20], [S23], [S26]
Not specified	4%	[S12]
Evidence		
Source code	12%	[S1], [S5], [S20]
Case studies	27%	[S1], [S3], [S5], [S10], [S11], [S13], [S20]
Experimental results	8%	[S5], [S10]
Solutions	0%	
Questionnaires	0%	
Transcripts	12%	[S11], [S13], [S20]
Additional statistics	4%	[S1]
Other	8%	[S5], [S10]
Statistical tests		
Yes	58%	
Pairwise comparison	50%	[S1], [S3], [S6], [S8], [S10], [S12], [S17], [S18], [S20], [S22], [S23], [S25], [S26]
Multiple comparison	19%	[S4], [S5], [S6], [S10], [S12]
Effect size measurement	12%	[S1], [S17], [S26]
No	42%	[S2], [S7], [S9], [S11], [S13], [S14], [S15], [S16], [S19], [S21], [S24]

TABLE 14
Characteristics of the participants in empirical studies (user's profile)

Number of participants	
1 participant	
Between 2 and 10	[S1], [S9], [S10], [S13], [S14], [S16], [S18], [S21], [S24]
Between 11 and 20	[S2], [S8], [S20], [S22], [S23], [S25]
More than 20	[S6], [S17]
Participant position	
Engineer (industry)	[S1], [S2], [S13], [S18], [S22], [S23]
PhD academia staff	[S9], [S10], [S22]
Postgraduate student	[S9], [S17], [S20], [S21], [S22], [S23], [S24]
Undergraduate student	[S10], [S20], [S22], [S25]
Other	[S6], [S8]
Not specified	[S14], [S16]
Participant expertise (in years)	
Less than 5	[S2], [S10], [S17], [S18], [S22], [S23]
Between 5 and 10	[S1], [S2], [S10], [S13], [S18], [S22], [S23]
Between 11 and 20	[S2], [S10], [S13], [S18], [S22], [S23]
More than 21	[S10]
Not specified	[S6], [S8], [S9], [S20], [S21], [S24], [S25]

TABLE 15
Statistical tests applied in iSBSE

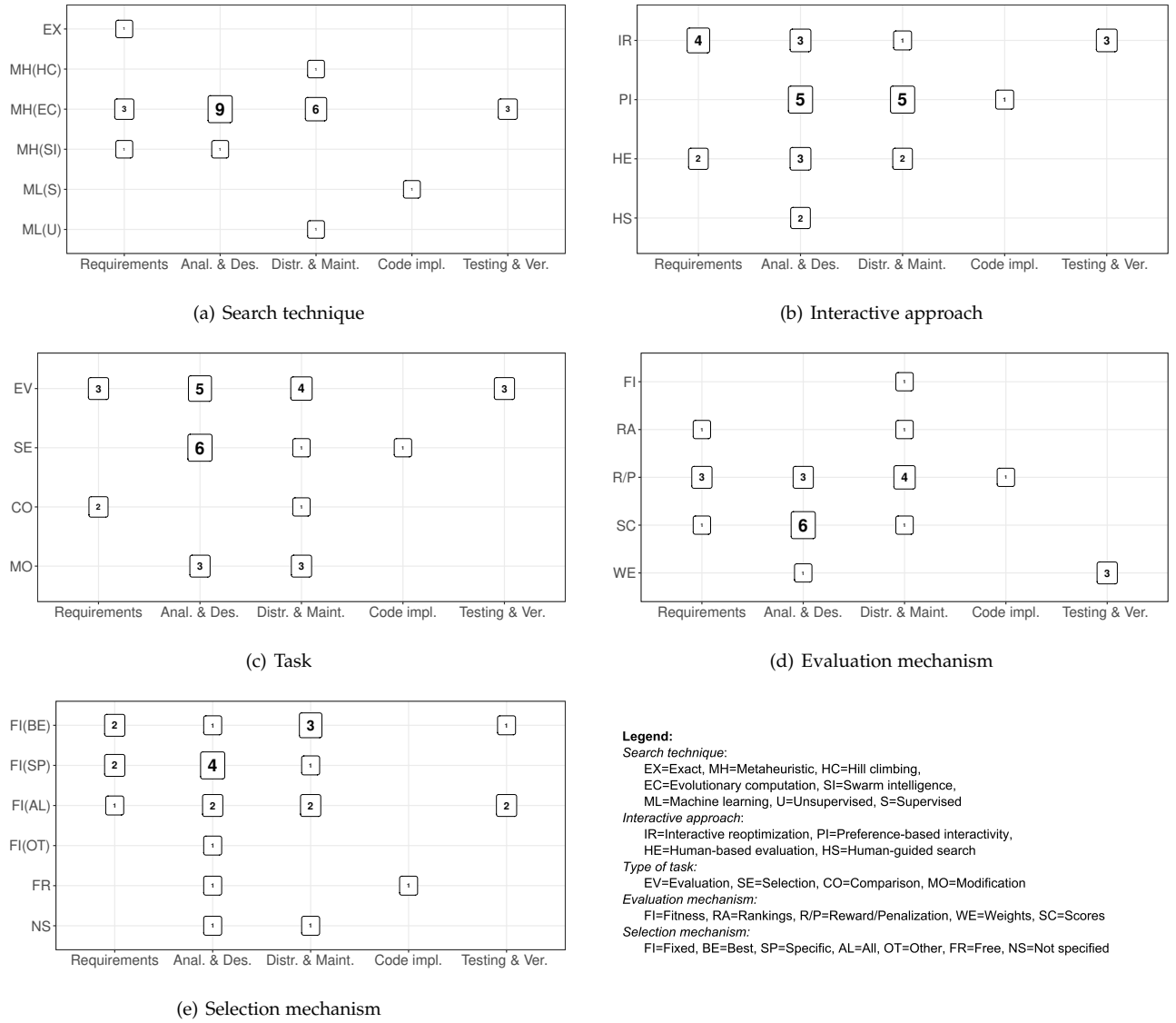
Pairwise comparison	
Student's t-test	[S8], [S25]
Wilcoxon Mann-Whitney	[S1], [S3], [S6], [S17], [S18], [S20], [S22] [S8], [S10], [S12], [S23], [S26]
Multiple comparison	
ANOVA	[S4], [S5]
Friedman	[S10]
Kruskal-Wallis	[S6], [S12]
Effect size measurement	
Cliff's Delta	[S1], [S17]
Vargha and Delaney	[S26]

disagreements to be solved.

Focusing on selection mechanisms for the presentation of candidate solutions (see Fig. 2(e)), it is apparent that showing a fixed numbers of solutions – especially BE (best) and AL (all) solutions – is preferred in almost all problem domains. Showing specific solutions might be of highest interest to the user, i.e. FI(SP) (fixed, specific). It was considered in the earliest stages of iSBSE to face requirements or design problems [S5], [S10], [S11], [S12]. However, in the last few years studies seem to be more oriented towards BE and AL, i.e. selecting the best solution [S2], [S20] or all of them [S1], [S17], independently of the problem domain.

Taking the extracted data results analysis shown in figures 2(a) to 2(e) overall, it is clear that some categories of search types and interactive approaches do occur for some problem formulations in the primary studies more

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSE.2018.2803055, IEEE Transactions on Software Engineering



(a) Search technique

(b) Interactive approach

(c) Task

(d) Evaluation mechanism

(e) Selection mechanism

Fig. 2. Relation between Software Engineering phases and other categories

frequently than others. To summarize and highlight these with respect to the following problem formulations:

- *Requirements*: *Interactive reoptimization* is the most frequently occurring interactive algorithm, and with *reward/penalization* the most frequent problem evaluation mechanism.
- *Design*: *Preference-based interactivity* is the most frequently occurring interactive algorithm, with *selection* and *evaluation* the most frequent interactive tasks for the user, together with *scores* as the the most frequent solution evaluation mechanism.
- *Maintenance*: *Preference-based interactivity* is again the most frequently occurring interactive algorithm, and *reward/penalization* the most frequent solution evaluation technique.

For code and testing, because of the paucity of studies, it seems too early to reach meaningful conclusions and make suggestions about how interactivity is being considered within these fields.

6.2 Search Technique and Interactive Approach

Focusing on the search algorithms, there is no clear trend regarding the application of a specific interactive model. Some recent studies apply human-based evaluation [S1], [S6]. Other approaches like preference-based interactivity and interactive reoptimization can be also found over the years, the latter being applied by an evolutionary algorithm for the first time in 2012 [S10]. Likewise, ant colony optimization was firstly applied following the precepts of human-based evaluation [S8], whereas interactive reoptimization has been considered more recently in 2016 [S2]. In addition, we note that machine learning does not directly work on candidate solutions as SBSE does, and the problem is not defined in terms of a fitness function. Consequently, the studies within

the ML paradigm fall into the same interactive approach (PI).

By a wide margin, the most frequently occurring interactive task is for evaluation of solutions and evolutionary computation (with 13 studies). Notice that the ultimate goal of user interaction is to provide some kind of evaluation, at least indirectly. This can be observed for swarm intelligence too, given that the two studies using the ACO paradigm [S2], [S8] focus their interaction on the evaluation phase. However, [S8] combines this task with the selection and modification of solutions. The latter also appears in combination with evaluation in three evolutionary-based studies [S9], [S23], [S26], which suggests that this decision is mostly motivated by the specific requirements of the problem domain. It is worth noting that both primary studies based on machine learning [S16], [S25] propose that the human participates in selecting input data to enable the subsequent mining process to produce more accurate classification models. However, how the user could help to improve other phases of the learning process (such as selecting features of interest or interactively supervising the construction of the classifier) remains as yet unexplored.

In analyzing how frequently each search technique applies an evaluation or a selection mechanism, little or no correspondence is immediately apparent. More specifically, it seems that there exists an independence between the underlying search technique (acting as a 'search engine') and the implementation of the user's interactive experience, at least in terms of interactive task, evaluation mechanism, and selection of candidate solutions for presentation and visualization. This is a valuable aspect of iSBSE, since it offers the flexibility to incorporate a customized interactive experience related to the characteristics of the problem domain independently from the underlying 'search engine'.

7 Open Issues and Future Trends

Taking the results of the systematic literature review presented in previous sections overall, it is evident that iSBSE has attracted significant research attention over recent years and substantive progress has been made. However, the review also reveals a number of challenges. In the following, we comment on some open issues and also speculate on future trends, as a response to RQ4.

Distance between the 'Optimizer' and the Software Engineer: In a real industrial setting, most probably those researchers who design and develop iSBSE software systems may not be the same persons acting as SE stakeholders to provide the domain-specific requirements, and subsequently use and interact with the iSBSE software system. If so, several issues might arise. For example, iSBSE requirements elicitation and analysis is crucial particularly with regard to the required interactive user experience. In addition, search algorithms chosen for implementation may not always be entirely appropriate for the given software engineering problem domain and development process. We speculate that it would be highly advantageous to engage with SE stakeholders, possibly by means of pilot studies, to validate and refine the user interactive experience, and its relationship to the search 'optimizer', prior to conducting experimental investigations.

User Trust and Acceptance: Achieving user trust and acceptance remains an open challenge for the field of iSBSE. The issue of 'automation bias' [15] is highly relevant, as is user complacency [16], [17]. Lyell and Coiera [15] suggest that strategies to minimize automation bias might focus on cognitive load reduction. The suggestion is echoed by Jackson [56] who points out that supporting human understanding is an important role for automation in software engineering, and it seems possible that wherever there is a lack of understanding, a lack of user trust and acceptance might follow. In addressing this challenge, it is noteworthy that Marculescu et al. [S17] report the use of the NASA Task Load Index (NASA-TLX), a subjective workload assessment tool (see Hart and Staveland [57]). It is also noteworthy that qualitative methods such as Grounded Theory [58] may have a role to play in the analysis of free format questionnaire data. Unlike quantitative approaches, where typically a pre-formed theory is proposed (possibly as one or more hypotheses) and experiments are designed to collect data to prove or refute hypotheses, Grounded Theory starts without a pre-formed theory and through repeated data collection and analysis, allows a theory to emerge from the data [59], [60]. As part of efforts to achieve user trust and acceptance in iSBSE, we would encourage consideration of

- techniques such as cognitive load monitoring and qualitative methods to examine the interactive user experience, and
- effective mechanisms to hide the underlying complexity of search techniques, such as reducing the number of parameters required of the user.

Role of the User: The nature of user participation in iSBSE also remains an open issue for the field. As observed in Section 5.3, it is possible that less rigid interaction mechanisms beyond evaluation of solutions, including direct manipulation of candidate solutions and the search process itself, may prove beneficial. However, previous work from other fields may offer possibilities to address this. For example, Kosorukoff [61] proposes a human-based genetic algorithm, wherein all primary genetic operators are delegated to the human. Kosorukoff asserts that the advantage of such an approach is its ability to not only facilitate the direct evaluation of individuals, but also to identify a good representation for them. In a further example, Bush and Sayama [62] propose hyperinteractive evolutionary computation (HIEC), in which the user proactively chooses when and how each evolutionary operator is applied. In human-subject experiments comparing HIEC with interactive evolutionary computing, Bush and Sayama report that HIEC facilitated a *"more controllable, more fun and more satisfying"* user experience. In a more recent study, Byrne et al. [63] propose an interactive methodology to enable the user to directly interact with the encoding of the candidate solutions they find aesthetically pleasing, and assert that *"broadening interaction beyond simple evaluation increases the amount of feedback and bias a user can apply to the search."* Given the crucial role of the user in iSBSE, we would also encourage a flexible and rich role for the user, possibly by drawing inspiration from the wider interactive metaheuristic search research community.

Assessment of Solution Fitness: User evaluation of solution fitness remains an open challenge to the iSBSE community, as software engineers can struggle to bring their 'hidden' knowledge to bear during interaction. 'Hidden' knowledge can relate to user qualitative evaluation of aesthetic aspects, such as the visual appeal of graphical user interfaces, or software design elegance. For example, Quiroz et al. [S14] compute the fitness of a graphical user interface using a weighted combination of user aesthetic evaluation and interface design guidelines. Troiano et al. [S6] search for a good compromise between interface aesthetics and accessible color schemes, taking account of two different fitness factors: an objective measure of color distances and contrast ratios found in the user interface and a subjective aesthetic score given by user. In a further example of the use of 'hidden' knowledge [S10], user measures relating to aspects of software design elegance (e.g. symmetry) are integrated with more established measures of coupling and cohesion during search to achieve more 'human-looking' solutions. Moreover, the notion of distinguishing user preferences as either hidden or implicit (as opposed to explicit) is explored by Aljawadeh et al. [14]. Results of this review suggest that user preferences can be short term or long term, which is consistent with the concept of a dynamic and flexible user experience recommended by Aleti et al. [26]. In addition, user fatigue is a crucial factor for the assessment of solution fitness [2]. Some tactics to reduce its effects have been offered by Shackelford [19] and these have been extended in the 'interactive solution presentation' metaheuristic design pattern [64]. Furthermore, reflecting the SBSE-related trend to Pareto-optimal many/multi-objective approaches [30], human solution fitness assessment and dynamic trade-off analysis present a challenge for many/multi-objective search algorithms. Current approaches are mostly focused on the dynamic adjustment of objectives during search [S9], [S17], which simplifies the problem into a single-objective one. Only a few studies maintain the original problem formulation, trying to infer user's judgment from actions such as the prioritization of non-dominated solutions [S23] or by means of sub-populations whose size varies with the relative importance of each objective as perceived by the user [S10]. It is interesting to observe that although many reports comparing multi/many objective Pareto-optimal approaches have appeared in the SBSE field [65], [66], [67], reports of dynamic, "in-the-loop" adjustment of the relative user importance of objective trade-offs for these approaches are less readily available, and we speculate that this might seem a fruitful avenue for future research.

Choice of Search Technique: As revealed in Section 5.2, evolutionary computation has been chosen as a search technique across a variety of SE problem formulations. However, it is interesting to note that studies comparing the relative performance of various metaheuristics generally, and iSBSE specifically, are not abundant in the literature, especially with regard to multi-objective optimization. Having said that, in a recent study, Piotrowski et al. [68] compare the relative performance of 33 metaheuristic techniques on 22 numerical real-world problems across a range of scientific domains. Results obtained would appear to be broadly consistent with those previously observed by Simons and Smith [69], who conducted a study comparing the performance of greedy local search, an evolutionary algorithm and ACO for search-based software design. Piotrowski et al. report that *"Particle Swarm Optimization algorithms and some new types of metaheuristics perform relatively better when the number of allowed function calls is low, whereas Differential Evolution and Genetic Algorithms perform better relative to other algorithms when the computational budget is large"*. Such findings resonate with the need to achieve effective search performance quickly in iSBSE, wherein reducing waiting times might help to mitigate the sensation of fatigue perceived by the user. While the findings of these two studies cannot be considered to be a comprehensive and complete picture of this open issue, they do however suggest that the use of swarm intelligence is worthy of consideration when choosing a search technique for iSBSE, particularly for multi-objective scenarios.

Available Frameworks and Implementations: Following the choice of search technique, a further open issue relates to the extent to which the availability of development frameworks for metaheuristic search might influence the choice of the search technique. Mature frameworks for the development of evolutionary computation approaches are readily available (e.g. jMetal [70], JCLEC [71]) and it is possible that this availability might contribute to the bias towards evolutionary algorithms found in the review results. It also seems possible that iSBSE researchers might not necessarily have appropriate expertise in optimization or search algorithms, and so seek to build on advances in the field by utilizing benchmarked implementations of algorithms. It is interesting to speculate that comparable studies (e.g. of similar algorithm implementations) could be a criterion for researchers in their choice of a search technique.

Experimental Validity and Empirical Studies: As discussed in Section 5.4, experimental evaluation of iSBSE research with respect to both research and empirical industrial contexts remains an on-going challenge. Given the emerging state of the iSBSE field, there appears to be a general lack of readily agreed and available benchmark case studies and standards. Considering the presence of human interaction with search, statistical analysis is appropriate, although it is recognized that this might be difficult if the sample size (in terms of number of participants and problem instances) is small. To address this, we note that Arcuri and Briand offer a helpful guide to statistical tests for assessing randomized algorithms in software engineering [47], and Neumann et al. suggest a useful executable experimental template pattern for the systematic comparison of metaheuristics [72]. Furthermore, important aspects like reproducibility are also related to additional materials being made available, allowing experiments and results to be contrasted. However, the results of this review reveal that additional experimental materials are made available in just 27% of the primary studies, and we would encourage authors to consider this, respecting ethical and confidentiality issues where appropriate. Notice that the evaluation of interactive sessions is essential to give insights about the effectiveness of the interaction with the software engineer. However, the lack of commonly accepted guidelines to design and conduct experiments in iSBSE makes difficult the comparison of different proposals. In this context, the use of common

problem instances and mechanisms to collect and process data, as well as unifying the creation of questionnaires would be highly recommended. It also seems likely that, with regard to industrial case studies in iSBSE, it remains a logistic challenge to recruit sufficient participants of sufficient expertise to be representative of some subset of the software engineering practitioner community. Lastly, in the longer term, we suggest that assessing the economic benefit of iSBSE (as suggested by Aleti et al. [26] for the field of Operations Research), may be important for promoting future industrial adoption.

Combining other Artificial Intelligence Techniques with iSBSE: We also speculate that one future trend lies in the combination of other artificial intelligence techniques, e.g. machine learning, with iSBSE, to learn implicit user preferences and thus help reduce user fatigue and improve the interactive user experience. It is interesting to note that in one of the primary studies, in addressing the next release problem, Araújo et al. [S1] report that the expertise of the decision maker is more effectively incorporated when machine learning techniques integrate tacit human knowledge with the optimization process. A further example of research combines machine learning and SBSE, where Amal et al. [S22] report the application of machine learning to classify ill-defined, implicit fitness functions using a case study on search based software refactoring. In a different (non-interactive) approach, Aleti et al. [73] incorporate a probabilistic Bayesian heuristic during search-based optimization of component deployment in embedded software systems. This is achieved by calculating the *a posteriori* probability that a particular component/host assignment constitutes a good outcome, which is then used to identify high quality deployed architectures, given a number of observations during search. These examples suggest that learning from the optimization process can provide valuable knowledge in SBSE, but further research is required into how reuse of such knowledge might be exploited when addressing related problems. We speculate that future research in iSBSE might investigate the opportunities that artificial intelligence can bring beyond the partial substitution of the human evaluation.

After an in-depth analysis of iSBSE, we have also observed some *gaps in the field* that might turn into research opportunities. As revealed in Section 5.1, results of the review indicate that **some lifecycle project phases appear to be as yet unexplored** in the primary studies. The lack of iSBSE studies in the areas of source code implementation and project management would seem to indicate a gap in the field. As noted previously, the deficiency of research studies into interactive search-based project management is interesting, considering its analogy to requirements prioritization, and the fact that others have investigated interactive search-based project management scheduling approaches more generally [19], [74]. In addition, we speculate that **incorporating expert knowledge into search** in iSBSE represents a gap in the field. As is typical in metaheuristic search, sources in the review initialize populations of solution individuals either at random or according to domain-specific heuristics. However, the reuse of expert knowledge (e.g. patterns, benchmarks etc.) to initialize populations might offer possibilities. We also observe that while individual phases of development are well supported by iSBSE, each is presented as if isolated from others, with different solution representations and search operators. Knowledge may be generated by search at one phase, but often discarded thereafter. We conjecture that a fruitful focus of research might address the **bringing together these isolated search elements in a development chain**, aligned to the software engineering development process.

8 Threats to Validity

Threats to the validity of this systematic review mainly relate to factors that might affect outcomes without the researchers' knowledge. Such threats center principally on the selection of primary studies and the process of extracting the appropriate data from the primary studies.

Regarding the selection of primary studies for the review, we describe in Section 3.1 the steps taken to justify their systematic and unbiased selection. To ensure relevant studies were located, we formulated a list of search strings derived from terms from a variety of application domains and search techniques. We conducted multiple iterative pilot searches to validate the adequacy and effectiveness of the search strings, looking for publications in the broadest sense. When required, a search string was adapted to the specific query language used by a particular engine or database. A yield of 653 sources returned suggests a level of confidence in the breadth of search. Having identified relevant sources using the search strings, we drafted and composed sets of inclusion and exclusion criteria to specifically identify those sources related to interactive SBSE. In Section 3.2, we state the inclusion and exclusion criteria to precisely describe the scope of the interactive role of the software engineer in SBSE. After applying inclusion and exclusion criteria, a resulting focus on 26 remaining primary sources out of the 669 originally returned by search also suggests a level of confidence in the inclusion and exclusion process.

With regard to the process of accurate and objective data extraction from the primary sources, we first formulated an appropriate classification scheme (see Section 3.4), which precisely describes the data items to be extracted. The classification scheme was derived by reference to other rigorous classifications of interactive metaheuristic search previously presented in other fields, and the specific context of SBSE. Secondly, each primary source was independently scrutinized by at least two reviewers, and in the case of any disagreements occurring, a third reviewer's assessment was brought to bear as arbiter. To prevent any potential conflict of interest, reviewers did not participate in data extraction where they were also an author of the primary source. Finally, we believe that the data extraction review process itself provided a thorough and robust validation of the classification scheme, which we hope may itself prove a useful contribution and starting point for other researchers in the field.

9 Conclusion

The design, development and maintenance of complex software systems can be challenging for the software engineer

to perform. The application of computational optimization methods has been shown to be an efficient mechanism to support the engineer in different software engineering tasks during development. However, there are still many problems where human intervention is crucial for solution evaluation and indication of preferences to the algorithm about how a given situation should be solved. In this work, we present the results of a systematic literature review into interaction within search-based software engineering. Such interaction occurs when the software engineer provides their implicit and/or explicit feedback during the search process, which significantly modifies the results and performance of the search.

To enable analysis of the primary studies, we formulate a classification scheme focused on four key aspects of iSBSE, i.e. the problem formulation, search technique, interactive approach, and the empirical framework used in experiments. Using this classification scheme, we analyzed the primary sources and present the results in manner that enables researchers to relate their work to the current body of knowledge and recognize future trends. We organized our results into tables and graphics, aiming to enable knowledge transfer for industrial practitioners, and across research communities whose focus may lie in distinct phases across the software development lifecycle. We note that research into iSBSE is multi-disciplinary in the sense of drawing on three areas, i.e. human-computer interaction, artificial intelligence (e.g. metaheuristic search, machine learning etc.), and software engineering. It seems unlikely that researchers might possess expertise in all three areas, and so this review helps to bridge the gap between the discipline-based communities and promote inter-disciplinary knowledge transfer.

The review and analysis based on the classification scheme highlights some open research issues for the field of iSBSE. The role of the user during interaction with search remains a challenge for researchers, particularly with regard to provision of a flexible and dynamically adapting user experience to promote user trust and acceptance of the search results obtained. We note that this is non-trivial to implement, and so conclude that it is crucial to take account of user requirements, software engineering context and human factors in the development of an iSBSE system. The generation and management of knowledge during iSBSE also represents an interesting and fruitful research trend for iSBSE. Currently, much knowledge about the problem formulation and candidate solutions is generated during interactive search, but is seldom retained for longer term use. We also conclude that future research might fruitfully be directed towards the acquisition and exploitation of knowledge generated during interactive search.

Finally, the results of this systematic review will enable the advancement of the iSBSE research community as the area continues to grow, and hopefully the classification scheme presented herein might form the beginning of a methodological approach for the design of iSBSE systems.

ACKNOWLEDGMENTS

This Work is supported in part by the Spanish Ministry of Economy and Competitiveness, projects TIN2014-55252-P and TIN2015-71841-REDT, the Spanish Ministry of Education under the FPU program (FPU13/01466), and FEDER funds.

PRIMARY STUDIES

[S1] A. A. Araújo, M. Paixao, I. Yeltsin, A. Dantas, and J. Souza, "An Architecture based on interactive optimization and machine learning applied to the next release problem," *Automat. Softw. Eng.*, vol. 24, no. 3, pp. 623–671, 2017.

[S2] T. do Nascimento Ferreira, A. A. Araújo, A. D. B. Neto, and J. T. de Souza, "Incorporating user preferences in ant colony optimization for the next release problem," *Appl. Soft Comput.*, vol. 49, pp. 1283–1296, 2016.

[S3] A. Dantas, I. Yeltsin, A. A. Araújo, and J. Souza, "Interactive Software Release Planning with Preferences Base," in *Proc. 7th Int. Symp. Search-Based Software Engineering*, 2015, pp. 341–346.

[S4] P. Tonella, A. Susi, and F. Palma, "Interactive requirements prioritization using a genetic algorithm," *Inf. Softw. Technol.*, vol. 55, no. 1, pp. 173–187, 2013.

[S5] F. Palma, A. Susi, and P. Tonella, "Using an SMT Solver for Interactive Requirements Prioritization," in *Proc. 19th ACM SIGSOFT Symp. / 13th Eur. Conf. Found. Softw. Eng.*, 2011, pp. 48–58.

[S6] L. Troiano, C. Birtolo, and R. Armenise, "A validation study regarding a generative approach in choosing appropriate colors for impaired users," *SpringerPlus*, vol. 5, no. 1, p. 1090, 2016.

[S7] A. E. El Yamany, M. Shaheen, and A. S. Sayyad, "OPTI-SELECT: An Interactive Tool for User-in-the-loop Feature Selection in Software Product Lines," in *Proc. 18th Int. Software Product Line Conf.: Companion Vol. for Workshops, Demonstrations and Tools - Vol. 2*, 2014, pp. 126–129.

[S8] C. L. Simons, J. Smith, and P. White, "Interactive ant colony optimization (iACO) for early lifecycle software design," *Swarm Intell.*, vol. 8, no. 2, pp. 139–157, 2014.

[S9] S. Vathsavayi, H. Hadaytullah, and K. Koskimies, "Interleaving human and search-based software architecture design," *Proc. Estonian Academy of Sciences*, vol. 62, no. 1, pp. 16–26, 2013.

[S10] C. L. Simons and I. C. Parmee, "Elegant Object-Oriented Software Design via Interactive, Evolutionary Computation," *IEEE Trans. Syst., Man, Cybern. C, Applications and Reviews*, vol. 42, no. 6, pp. 1797–1805, 2012.

[S11] C. L. Simons, I. C. Parmee, and R. Gwynllyw, "Interactive, Evolutionary Search in Upstream Object-Oriented Class Design," *IEEE Trans. Softw. Eng.*, vol. 36, no. 6, pp. 798–816, 2010.

[S12] C. Birtolo, P. Pagano, and L. Troiano, "Evolving colors in user interfaces by interactive genetic algorithm," in *Proc. World Congr. Nature Biologically Inspired Computing*, 2009, pp. 349–355.

[S13] C. L. Simons and I. C. Parmee, "An empirical investigation of search-based computational support for conceptual software engineering design," in *Proc. IEEE Int. Conf. Syst., Man, Cybern.*, 2009, pp. 2503–2508.

[S14] J. C. Quiroz, S. J. Louis, A. Shankar, and S. M. Dascalu, "Interactive Genetic Algorithms for User Interface Design," in *Proc. IEEE Congr. on Evolutionary Computation*, 2007, pp. 1366–1373.

[S15] N. Monmarché, G. Nocent, M. Slimane, G. Venturini, and P. Santini, "Imagine: a tool for generating html style sheets with an interactive genetic algorithm based on genes frequencies," in *Proc. IEEE Int. Conf. Syst., Man, Cybern.*, vol. 3, 1999, pp. 640–645.

[S16] S. Axelsson, D. Baca, R. Feldt, D. Sidlauskas, and D. Kacan, "Detecting Defects with an Interactive Code Review Tool Based on Visualisation and Machine Learning," in *Proc. 21st Int. Conf. Softw. Eng. Knowl. Eng.*, 2009, pp. 412–417.

[S17] B. Marculescu, S. Poulding, R. Feldt, K. Petersen, and R. Torkar, "Tester interactivity makes a difference in search-based software testing: A controlled experiment," *Inf. Softw. Technol.*, vol. 78, pp. 66–82, 2016.

[S18] B. Marculescu, R. Feldt, R. Torkar, and S. Poulding, "An initial industrial evaluation of interactive search-based testing for embedded software," *Appl. Soft Comput.*, vol. 29, pp. 26–39, 2015.

[S19] B. Marculescu, R. Feldt, and R. Torkar, "Objective Re-weighting to Guide an Interactive Search Based Software Testing System," in *Proc. 12th Int. Conf. Machine Learning and Applications*, vol. 2, Dec 2013, pp. 102–107.

[S20] Y. Lin, X. Peng, Y. Cai, D. Dig, D. Zheng, and W. Zhao, "Interactive and guided architectural refactoring with search-based recommendation," in *Proc. 24th ACM SIGSOFT Int. Symp. Found. Softw. Eng.*, 2016, pp. 535–546.

[S21] M. W. Mkaouer, "Interactive Code Smells Detection: An Initial Investigation," in *Proc. 8th Int. Symp. Search-Based Software Engineering*, 2016, pp. 281–287.
[S22] B. Amal, M. Kessentini, S. Bechikh, J. Dea, and L. B. Said, "On the Use of Machine Learning and Search-Based Software Engineering for Ill-Defined Fitness Function: A Case Study on Software Refactoring," in *Proc. 6th Int. Symp. Search Based Software Engineering*, 2014, pp. 31–45.
[S23] M. W. Mkaouer, M. Kessentini, S. Bechikh, K. Deb, and M. Ó Cinnéide, "Recommendation System for Software Refactoring Using Innovization and Interactive Dynamic Optimization," in *Proc. 29th ACM/IEEE Int. Conf. Automat. Softw. Eng.* ACM, 2014, pp. 331–336.
[S24] A. Ghannem, G. El Boussaidi, and M. Kessentini, "Model Refactoring Using Interactive Genetic Algorithm," in *Proc. 5th Int. Symp. Search Based Software Engineering*, 2013, pp. 96–110.
[S25] J. Wang, X. Peng, Z. Xing, and W. Zhao, "Improving Feature Location Practice with Multi-faceted Interactive Exploration," in *Proc. Int. Conf. Softw. Eng.*, 2013, pp. 762–771.
[S26] G. Bavota, F. Carnevale, A. De Lucia, M. Di Penta, and R. Oliveto, "Putting the Developer in-the-Loop: An Interactive GA for Software Re-modularization," in *Proc. 4th Int. Symp. Search Based Software Engineering*, 2012, pp. 75–89.

REFERENCES

[1] M. Harman and B. F. Jones, "Search-based software engineering," *Inf. Softw. Technol.*, vol. 43, no. 14, pp. 833–839, 2001.
[2] M. Harman, S. A. Mansouri, and Y. Zhang, "Search-based software engineering: Trends, techniques and applications," *ACM Comput. Surv.*, vol. 45, no. 1, p. 11, 2012.
[3] D. Meignan, S. Knust, J.-M. Frayret, G. Pesant, and N. Gaud, "A review and taxonomy of interactive optimization methods in operations research," *ACM Trans. Interact. Intell. Syst.*, vol. 5, no. 3, p. 17, 2015.
[4] B. Curtis, H. Krasner, and N. Iscoe, "A field study of the software design process for large systems," *Communications of the ACM*, vol. 31, no. 11, pp. 1268–1287, 1988.
[5] R. Guindon, "Designing the design process: Exploiting opportunistic thoughts," *Human-Comp. Interact.*, vol. 5, no. 2, pp. 305–344, 1990.
[6] A. E. Eiben and J. E. Smith, *Introduction to evolutionary computing*, 2nd ed. Springer, 2015.
[7] G. Klien, D. D. Woods, J. M. Bradshaw, R. R. Hoffman, and P. J. Feltovich, "Ten challenges for making automation a "team player" in joint human-agent activity," *IEEE Intell. Syst.*, vol. 19, no. 6, pp. 91–95, 2004.
[8] D. Jones, "Programming using genetic algorithms: isnt that what humans already do," accessed 8 November 2016. [Online]. Available: http://shape-of-code.coding-guidelines.com/2013/10/18/programming-using-genetic-algorithms-isnt-that-what-humans-already-do/
[9] C. Le Goues, S. Forrest, and W. Weimer, "Current challenges in automatic software repair," *Software Qual. J.*, vol. 21, no. 3, pp. 421–443, 2013.
[10] R. Dawkins, "The blind watchmaker," *Harlow: Longman*, 1986.
[11] K. Sims, "Evolving virtual creatures," in *Proc. 21st Ann. Conf. Computer graphics and interactive techniques*, 1994, pp. 15–22.
[12] H. Takagi, "Interactive evolutionary computation: Fusion of the capabilities of EC optimization and human evaluation," *Proc. IEEE*, vol. 89, no. 9, pp. 1275–1296, 2001.
[13] J. Branke, K. Deb, K. Miettinen, and R. Slowiński, *Multiobjective optimization: Interactive and evolutionary approaches*. Springer, 2008, vol. LCNS 5252.
[14] H. J. Aljawawdeh, C. L. Simons, and M. Odeh, "Metaheuristic Design Pattern: Preference," in *Proc. Companion Publication Ann. Conf. Genetic and Evolutionary Computation*, 2015, pp. 1257–1260.
[15] D. Lyell and E. Coiera, "Automation bias and verification complexity: a systematic review," *J. Am. Med. Inform. Assoc.*, p. ocw105, 2016.
[16] R. Parasuraman and V. Riley, "Humans and automation: Use, misuse, disuse, abuse," *Hum. Fac.: J. Hum. Fac. Erg. Soc.*, vol. 39, no. 2, pp. 230–253, 1997.
[17] L. J. Skitka, K. L. Mosier, and M. Burdick, "Does automation bias decision-making?" *Int. J. Hum.-Comp. Stud.*, vol. 51, no. 5, pp. 991–1006, 1999.
[18] J. E. Bahner, A.-D. Hüper, and D. Manzey, "Misuse of automated decision aids: Complacency, automation bias and the impact of training experience," *Int. J. Hum.-Comp. Stud.*, vol. 66, no. 9, pp. 688–699, 2008.
[19] M. Shackelford, "Implementation issues for an interactive evolutionary computation system," in *Proc. Companion Publication Ann. Conf. Genetic and Evolutionary Computation*, 2007, pp. 2933–2936.
[20] C. Alexander, *A Pattern Language: Towns, Buildings, Construction*. Oxford University Press, 1977.
[21] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design patterns: elements of reusable object-oriented software*. Addison-Wesley Pearson, 1995.
[22] M. Harman, "The role of artificial intelligence in software engineering," in *Proc. 1st Int. Workshop on Realizing AI Synergies in Software Engineering*. IEEE Press, 2012, pp. 1–6.
[23] B. A. Kitchenham, D. Budgen, and P. Brereton, *Evidence-Based Software Engineering and Systematic Reviews*. CRC Press, 2016.
[24] A. M. Pitangueira, R. S. P. Maciel, and M. Barros, "Software requirements selection and prioritization using SBSE approaches: A systematic review and mapping of the literature," *J. Syst. Softw.*, vol. 103, pp. 267–280, 2015.
[25] O. Räihä, "A survey on search-based software design," *Comput. Sci. Rev.*, vol. 4, no. 4, pp. 203–249, 2010.
[26] A. Aleti, B. Buhnova, L. Grunske, A. Koziolek, and I. Meedeniya, "Software Architecture Optimization Methods: A Systematic Literature Review," *IEEE Trans. Softw. Eng.*, vol. 39, no. 5, pp. 658–683, 2013.
[27] W. Afzal, R. Torkar, and R. Feldt, "A systematic review of search-based testing for non-functional system properties," *Inf. Softw. Technol.*, vol. 51, no. 6, pp. 957–976, 2009.
[28] G. Bavota, M. Di Penta, and R. Oliveto, *Search Based Software Maintenance: Methods and Tools*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2014, pp. 103–137.
[29] T. Mariani and S. R. Vergilio, "A systematic review on search-based refactoring," *Inf. Softw. Technol.*, vol. 83, pp. 14–34, 2017.
[30] A. S. Sayyad and H. Ammar, "Pareto-optimal search-based software engineering (POSBSE): A literature survey," in *Proc. 2nd Int. Workshop on Realizing Artificial Intelligence Synergies in Software Engineering*, 2013, pp. 21–27.
[31] M. Harman, "SBSE: Introduction, Motivation, Results and Directions (keynote)," in *6th Int. Symp. Search Based Software Engineering (SSBSE'14)*, 2014.
[32] M. Harman, Y. Jia, and Y. Zhang, "Achievements, Open Problems and Challenges for Search Based Software Testing," in *8th IEEE Int. Conf. Software Testing, Verification and Validation (ICST)*, 2015, pp. 1–12.
[33] C. Simons, J. Singer, and D. R. White, "Search-based refactoring: Metrics are not enough," in *Proc. 7th Int. Symp. Search-Based Software Engineering (SSBSE'15)*, 2015, pp. 47–61.
[34] G. R. Santhanam, "Qualitative optimization in software engineering: A short survey," *J. Syst. Softw.*, vol. 111, pp. 149–156, 2016.
[35] S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*. Prentice-Hall, 2009, ch. Part II: Problem Solving, pp. 64–202.
[36] C. Blum and A. Roli, "Metaheuristics in combinatorial optimization: Overview and conceptual comparison," *ACM Comput. Surv.*, vol. 35, no. 3, pp. 268–308, 2003.
[37] I. Boussaïd, J. Lepagnot, and P. Siarry, "A survey on optimization metaheuristics," *Inf. Sci.*, vol. 237, pp. 82–117, 2013.
[38] B. Li, J. Li, K. Tang, and X. Yao, "Many-Objective Evolutionary Algorithms: A Survey," *ACM Comput. Surv.*, vol. 48, no. 1, pp. 13:1–35, 2015.
[39] A. Zhou, B.-Y. Qu, H. Li, S.-Z. Zhao, P. N. Suganthan, and Q. Zhang, "Multiobjective evolutionary algorithms: A survey of the state of the art," *Swarm Evol. Comput.*, vol. 1, no. 1, pp. 32–49, 2011.
[40] I. C. Parmee, "Poor-Definition, Uncertainty, and Human Factors - Satisfying Multiple Objectives in Real-World Decision-Making Environments," in *Evolutionary Multi-Criterion Optimization*, 2001, pp. 52–66.
[41] M. Ware, E. Frank, G. Holmes, M. Hall, and I. H. Witten, "Interactive machine learning: letting users build classifiers," *Int. J. Hum.-Comp. Stud.*, vol. 55, no. 3, pp. 281–292, 2001.
[42] B. Kitchenham and S. Charters, "Guidelines for performing Systematic Literature Reviews in Software Engineering," Keele University and Durham University, Tech. Rep., 2007.
[43] N. Salleh, E. Mendes, and J. Grundy, "Empirical studies of pair programming for CS/SE teaching in higher education: A system-

atic literature review," *IEEE Trans. Softw. Eng.*, vol. 37, no. 4, pp. 509–525, 2011.

[44] A. M. Pitangueira, R. S. P. Maciel, M. de Oliveira Barros, and A. S. Andrade, "A systematic review of software requirements selection and prioritization using SBSE approaches," in *5th Int. Symp. Search Based Software Engineering*. Springer, 2013, pp. 188–208.

[45] R. S. Pressman and B. R. Maxim, *Software Engineering: A Practitioner's Approach*, 8th ed. McGraw-Hill Education, 2014.

[46] D. Zhang and J. J. Tsai, "Machine Learning and Software Engineering," *Software Qual. J.*, vol. 11, no. 2, pp. 87–119, 2003.

[47] A. Arcuri and L. Briand, "A Hitchhiker's guide to statistical tests for assessing randomized algorithms in software engineering," *Softw. Test. Verif. Rel.*, vol. 24, no. 3, pp. 219–250, 2014.

[48] A. A. Araújo and M. Paixão, "Machine Learning for User Modeling in an Interactive Genetic Algorithm for the Next Release Problem," in *6th Int. Symp. Search-Based Software Engineering*, 2014, pp. 228–233.

[49] B. Marculescu, R. Feldt, and R. Torkar, "Practitioner-Oriented Visualization in an Interactive Search-Based Software Test Creation Tool," in *20th Asia-Pacific Software Engineering Conf.*, vol. 2, 2013, pp. 87–92.

[50] ——, "A Concept for an Interactive Search-based Software Testing System," in *4th Int. Symp. Search Based Software Engineering*, 2012, pp. 273–278.

[51] P. Tonella, A. Susi, and F. Palma, "Using Interactive GA for Requirements Prioritization," in *2nd Int. Symp.Search Based Software Engineering*, 2010, pp. 57–66.

[52] L. Troiano, C. Birtolo, and G. Cirillo, "Interactive Genetic Algorithm for choosing suitable colors in User Interface," in *Proc. Learning and Intelligent OptimizatioN*, 2009, pp. 14–18.

[53] C. L. Simons and I. C. Parmee, "Agent-based Support for Interactive Search in Conceptual Software Engineering Design," in *Proc. 2008 Ann. Conf. Genetic and Evolutionary Computation*, 2008, pp. 1785–1786.

[54] C. Simons and I. Parmee, "User-centered, evolutionary search in conceptual software design," in *IEEE Congr. Evolutionary Computation*, 2008, pp. 869–876.

[55] N. Monmarché, G. Nocent, G. Venturini, and P. Santini, "On Generating HTML Style Sheets with an Interactive Genetic Algorithm Based on Gene Frequencies," in *4th Eur. Conf. Artificial Evolution*, 1999, pp. 99–110.

[56] M. Jackson, "Automated software engineering: supporting understanding," *Automat. Softw. Eng.*, vol. 15, no. 3, pp. 275–281, 2008.

[57] S. G. Hart and L. E. Staveland, "Development of NASA-TLX (Task Load Index): Results of empirical and theoretical research," *Adv. Psychol.*, vol. 52, pp. 139–183, 1988.

[58] A. Bryant and K. Charmaz, *The Sage Handbook of Grounded Theory*. Sage, 2007.

[59] M. Myers, "Qualitative research in information systems," *Manag. Inf. Syst. Q.*, vol. 21, no. 2, pp. 241–242, 1997.

[60] A. Strauss and J. Corbin, *Basics of Qualitative Research*, 3rd ed. Newbury Park, CA: Sage, 1990.

[61] A. Kosorukoff, "Human based genetic algorithm," in *IEEE Int. Conf. on Syst., Man, Cybern*, vol. 5, 2001, pp. 3464–3469.

[62] B. J. Bush and H. Sayama, "Hyperinteractive Evolutionary Computation," *IEEE Trans. Evol. Comput.*, vol. 15, no. 3, pp. 424–433, 2011.

[63] J. Byrne, E. Hemberg, M. ONeill, and A. Brabazon, "A methodology for user directed search in evolutionary design," *Genet. Program. Evol. M.*, vol. 14, no. 3, pp. 287–314, 2013.

[64] M. Shackelford and C. L. Simons, "Metaheuristic design pattern: interactive solution presentation," in *Proc. Companion Publication Ann. Conf. on Genetic and Evolutionary Computation*, 2014, pp. 1431–1434.

[65] A. Ramírez, J. R. Romero, and S. Ventura, "A comparative study of many-objective evolutionary algorithms for the discovery of software architectures," *Empir. Software Eng.*, vol. 21, no. 6, pp. 2546–2600, 2016.

[66] R. M. Hierons, M. Li, X. Liu, S. Segura, and W. Zheng, "SIP: Optimal Product Selection from Feature Models Using Many-Objective Evolutionary Optimization," *ACM Trans. Softw. Eng. Methodol.*, vol. 25, no. 2, pp. 17:1–17:39, 2016.

[67] M. W. Mkaouer, M. Kessentini, S. Bechikh, M. Ó Cinnéide, and K. Deb, "On the use of many quality attributes for software refactoring: a many-objective search-based software engineering approach," *Empir. Software Eng.*, vol. 21, no. 6, pp. 2503–2545, 2016.

[68] A. P. Piotrowski, M. J. Napiorkowski, J. J. Napiorkowski, and P. M. Rowinski, "Swarm intelligence and evolutionary algorithms: performance versus speed," *Inf. Sci.*, vol. 384, pp. 34–85, 2017.

[69] C. Simons and J. Smith, "A comparison of meta-heuristic search for interactive software design," *Soft Comput.*, vol. 17, no. 11, pp. 2147–2162, 2013.

[70] J. J. Durillo and A. J. Nebro, "jMetal: A java framework for multi-objective optimization," *Adv. Eng. Softw.*, vol. 42, no. 10, pp. 760–771, 2011.

[71] A. Ramírez, J. R. Romero, and S. Ventura, "An extensible jclec-based solution for the implementation of multi-objective evolutionary algorithms," in *Proc. Companion Publication Ann. Conf. Genetic and Evolutionary Computation*, 2015, pp. 1085–1092.

[72] G. Neumann, J. Swan, M. Harman, and J. A. Clark, "The executable experimental template pattern for the systematic comparison of metaheuristics," in *Proc. Companion Publication Ann. Conf. Genetic and Evolutionary Computation*, 2014, pp. 1427–1430.

[73] A. Aleti and I. Meedeniya, "Component deployment optimisation with bayesian learning," in *Proc. 14th Int. Symp. Component-Based Software Engineering*, 2011, pp. 11–20.

[74] M. Shackelford and D. Corne, "Collaborative evolutionary multi-project resource scheduling," in *Proc. IEEE Congr. Evolutionary Computation*, vol. 2, 2001, pp. 1131–1138.

Aurora Ramírez received a M.Sc. degree in Computer Science from the University of Córdoba, Spain, in 2012. Since 2012, she has been with the Knowledge Discovery and Intelligent Systems Research Laboratory of the University of Córdoba, where she is currently working towards obtaining a Ph.D. Her research interests include search-based software engineering, metaheuristics and data mining.

José Raúl Romero is Associate Professor at the University of Córdoba, Spain. Prior to receiving the PhD degree from the University of Málaga in 2007, he was an IT consultant. His current research interests include the the development of intelligent systems for industrial use, search-based software engineering and data science. He is a member of IEEE, ACM, and the Spanish Technical Normalization Committee AEN/CTN 71/SC7 of AENOR. He can also be reached at http://www.jrromero.net.

Christopher L. Simons is Senior Lecturer at the University of the West of England (UWE), Bristol, UK, from where he received the PhD degree in Interactive Evolutionary Computation. Prior to joining UWE in 2002, Chris practiced as a software engineer, architect and consultant. He is a member of the British Computer Society. His research interests center on interaction and mutual learning between software engineers and interactive computational intelligence.

8

Conference publications

8.1. International conferences and workshops

1. A. Ramírez, J.R. Romero, S. Ventura. *On the Performance of Multiple Objective Evolutionary Algorithms for Software Architecture Discovery.* In Proceedings of the 16th Genetic and Evolutionary Computation Conference (GECCO), pp. 1287-1294. Vancouver (Canada). July 2014. ACM. ***Best paper of the SBSE track and nominated to best paper of GECCO'14***. DOI: 10.1145/2576768.2598310. CORE(2014): A.

2. A. Ramírez, J.R. Romero, S. Ventura. *An Extensible JCLEC-based Solution for the Implementation of Multi-Objective Evolutionary Algorithms.* Evolutionary Computation Software Systems (EvoSoft) Workshop. In Proceedings of the Companion Publication of the 17th Genetic and Evolutionary Computation Conference (GECCO Companion), pp. 1085-1092. Madrid (Spain). July 2015. ACM. DOI: 10.1145/2739482.2768461. GGS(2015): Class 2 (A).

3. A. Ramírez, R. Barbudo, J.R. Romero, S. Ventura. *Memetic Algorithms for the Automatic Discovery of Software Architectures.* In Proceedings of the 16th International Conference on Intelligent Systems Design and Applications

(ISDA), vol. 557 AISC, pp. 437-447. Porto (Portugal). December 2016. Springer. DOI: 10.1007/978-3-319-53480-0_43. GGS(2015): Work in progress.

4. A. Ramírez, J.R. Romero, S. Ventura. *On the Effect of Local Search in the Multi-objective Evolutionary Discovery of Software Architectures*. In Proceedings of the Congress on Evolutionary Computation (CEC), pp. 2038-2045. San Sebastián (Spain). June 2017. IEEE. DOI: 10.1109/CEC.2017.7969551. GGS(2017): Class 2 (A-).

8.2. National conferences

1. A. Ramírez, J.R. Romero, S. Ventura. *Modelos metaheurísticos para el soporte a la decisión en la construcción de software*. Proceedings of the I Doctoral Consortium de la Sociedad de Ingeniería del Software y Tecnologías de Desarrollo de Software (SISTEDES), pp. 10-14. Cádiz (Spain). September 2014. ***Thesis proposal awarded a second prize.***

2. A. Ramírez, J.R. Romero, S. Ventura. *Análisis de la aplicabilidad de medidas software para el diseño semi-automático de arquitecturas*. Proceedings of the XIX Jornadas en Ingeniería del Software y Bases de Datos (JISBD), pp. 307-320. Cádiz (Spain). September 2014. Available from SISTEDES digital library: 11705/JISBD/2014/050.

3. A. Ramírez, J.R. Romero, S. Ventura. *Estudio preliminar del rendimiento de familias de algoritmos multiobjetivo en diseño arquitectónico*. Proceedings of the X Congreso Español sobre Metaheurísticas, Algoritmos Evolutivos y Bioinspirados (MAEB), pp. 173-180. Mérida (Spain). February 2015.

4. A. Ramírez, J.R. Romero, S. Ventura. *Interactividad en el descubrimiento evolutivo de arquitecturas*. Proceedings of the XX Jornadas en Ingeniería del Software y Bases de Datos (JISBD). Santander (Spain). September 2015. Available from SISTEDES digital library: 11705/JISBD/2015/029.

5. A. Ramírez, J.A. Molina, J.R. Romero, S. Ventura. *Estudio de mecanismos de hibridación para el descubrimiento evolutivo de arquitecturas*. Proceedings of the XXI Jornadas en Ingeniería del Software y Bases de Datos (JISBD), pp.

481-494. Salamanca (Spain). September 2016. Available from SISTEDES digital library: 11705/JISBD/2016/043.

6. J.A. Parejo, A. Ramírez, J.R. Romero, S. Segura, A. Ruiz-Cortés. *Configuración guiada por búsqueda de aplicaciones basadas en microservicios en la nube*. Proceedings of the XXI Jornadas en Ingeniería del Software y Bases de Datos (JISBD), pp. 499-502. Salamanca (Spain). September 2016. Available from SISTEDES digital library: 11705/JISBD/2016/045.

7. A. Ramírez, R. Barbudo, J.R. Romero, S. Ventura. *Herramienta basada en computación evolutiva interactiva para arquitectos software*. Proceedings of the XVII Conferencia de la Asociación Española para la Inteligencia Artificial - XI Congreso Español sobre Metaheurísticas, Algoritmos Evolutivos y Bioinspirados (CAEPIA-MAEB), pp. 387-396. Salamanca (Spain). September 2016.

8. A. Ramírez, J.R. Romero, S. Ventura. *Búsqueda coevolutiva interactiva aplicada al diseño de software*. Proceedings of the XXII Jornadas en Ingeniería del Software y Bases de Datos (JISBD). La Laguna, Tenerife (Spain). July 2017. Available from SISTEDES digital library: 11705/JISBD/2017/034.

9. A. Ramírez, J.R. Romero, S. Ventura. *API para el desarrollo de algoritmos interactivos en ingeniería del software basada en búsqueda*. XXIII Jornadas en Ingeniería del Software y Bases de Datos (JISBD). Sevilla (Spain). September 2018. Available from SISTEDES digital library: 11705/JISBD/2018/073.