Applied Computing Review

Sep. 2012, Vol. 12, No. 3

Frontmatter

Editors		**3**
SIGAPP Semi-Annual Report	S. Shin	**4**
A Message from the Editor	S. Shin	**5**
SAC 2013 Preview	H. Haddad	**6**

Selected Research Articles

Feature Space Optimization for Content-Based Image Retrieval	L. Avalhais, S. da Silva, J.F. Rodrigues Jr., A. Traina, and C. Traina Jr.	**7**
An Empirical Study on Clone Stability	M. Mondal, C. Roy, and K. Schneider	**20**
XFormsDB: An Extensible Web Application Framework Built upon Declarative W3C Standards	M. Laine, D. Shestakov, and P. Vuorimaa	**37**
Analysis of a Triploid Genetic Algorithm over Deceptive and Epistatic Landscapes	M. Li, S. Hill, and C. O'Riordan	**51**
SART: Speeding up Query Processing in Sensor Networks with an Autonomous Range Tree Structure	S. Sioutas, A. Panaretos, I. Karydis, D. Tsoumakos, G. Tzimas, and D. Tsolis	**60**

SIGAPP FY'12 Semi-Annual Report

March 2012 – August 2012
Sung Shin

Mission

To further the interests of the computing professionals engaged in the development of new computing applications and to transfer the capabilities of computing technology to new problem domains.

Officers

Chair	Sung Shin South Dakota State University, USA
Vice Chair	Richard Chbeir Bourgogne University, Dijon, France
Secretary	W. Eric Wong University of Texas at Dallas, USA
Treasurer	Lorie M. Liebrock New Mexico Institute of Mining and Technology, USA
Web Master	Hisham Haddad Kennesaw State University, USA
ACM Program Coordinator	Irene Frawley ACM HQ

Notice to Contributing Authors

By submitting your article for distribution in this Special Interest Group publication, you hereby grant to ACM the following non-exclusive, perpetual, worldwide rights:

- to publish in print on condition of acceptance by the editor
- to digitize and post your article in the electronic version of this publication
- to include the article in the ACM Digital Library and in any Digital Library related services
- to allow users to make a personal copy of the article for noncommercial, educational or research purposes

However, as a contributing author, you retain copyright to your article and ACM will refer requests for republication directly to you.

A Message from the Editor

I am excited to release the fall issue of Applied Computing Review (ACR). ACR has been published on a quarterly basis since last winter, and this issue includes five selected papers presented at the 2012 *ACM Symposium on Applied Computing (SAC)*. They have been expanded, revised, and reviewed for inclusion in ACR, and we are proud that all of them are high quality papers.

ACR provides you with a platform for sharing novel ideas among practitioners and professionals in various fields of applied computing. Moreover, we have provided excellent service to various technical communities and to the scientific computing society in a productive manner. We are working with the ACM SIG Governing Board to further expand SIGAPP by increasing membership. Also, we are working hard so that ACR can appear in *Science Citation Index (SCI)* in the near future. I would like to thank the authors for contributing the state-of-the-art methods in their research area. I am grateful to the highly qualified peer reviewers who coordinated an outstanding lineup of technical paper reviews. This issue of ACR couldn't have been published without significant efforts made by everyone, and I want to express my sincere gratitude to all of them.

In closing, I am pleased to tell you that the 28th SAC will be held next spring in Coimbra, Portugal. I would like many of you to join us and make the conference a great success. In addition, I hope you enjoy this issue of ACR as much as we do. Your continuous support and cooperation would be highly appreciated. Thank you.

Sincerely,

Sung Shin
Editor in Chief & Chair of ACM SIGAPP

Next Issue

The planned release for the next issue of ACR is December 2012.

SAC 2013 Preview

The 28th Annual edition of the ACM Symposium on Applied Computing (SAC) will be held at the Coimbra Institute of Engineering, March 18-22, 2013, Coimbra, Portugal. The conference Tutorials are planned for Monday, while the Posters and Technical program start on Tuesday.

Working with the local organizing committee, lead by Professor Nuno Ferreira of ISEC, we have selected designated hotels with special rates for SAC attendees. The hotel information will be posted on http://www.acm.org/conferences/sac/sac2013/ along with the reservation code for the special rates. The organizing committee recommends attendees to book their reservations at the designed hotels as soon as the room blocks are opened. The Conference will provide shuttle service between ISEC and the designated hotels. Detailed shuttle schedule will be posted on the website once finalized. Transfer service will also be provided from these hotels to the sites of SIGAPP Reception and the Banquet event. The daily lunches, coffee breaks, reception, and banquet dinner will be included in the registration fee. In addition, the local committee is organizing a number of excursions. Details will be posted on the website.

The conference will include the *Student Research Competition (SRC)* program. *SRC* is a new addition to SAC. It is designed to provide graduate students the opportunity to meet and exchange ideas with researchers and practitioners in their areas of interest. Active graduate students seeking feedback from the scientific community on their research ideas are invited to submit abstracts of their original un-published and in-progress research work in areas of experimental computing and application development related to SAC 2013 Tracks. Accepted research abstracts will be published in SAC CD Proceedings. Authors of accepted abstracts are eligible to apply for the SIGAPP *Student Travel Award Program (STAP)*. A designated committee will judge the presentations and select the top three winners for cash prizes ($500, $300, and $200, respectively). The winners will be recognized during the banquet event.

As the planning is underway, we are excited to have SAC in the historic city of Coimbra. We invite you to join us next March, meet other attendees, enjoy the conference programs, and have a pleasant stay in Coimbra. We hope to see you there.

On Behalf of SAC Steering Committee,

Hisham Haddad
Member of the Steering Committee
Member of SAC 2013 Organzing Commitee

Feature Space Optimization for Content-Based Image Retrieval

Letricia P. S. Avalhais, Sergio F. da Silva,
Jose F. Rodrigues Jr., Agma J. M. Traina and
Caetano Traina Jr.
{letricia, sergio, junio, agma, caetano}@icmc.usp.br
Institute of Mathematics and Computer Science
University of São Paulo
São Carlos, Brazil

ABSTRACT

Substantial benefits can be gained from effective Relevance Feedback techniques in content-based image retrieval. However, existing techniques are limited due to computational cost and/or by being restricted to linear transformations on the data. In this study we analyze the role of nonlinear transformations in relevance feedback. We present two promising Relevance Feedback methods based on Genetic Algorithms used to enhance the performance on the task of image retrieval according to the user's interests. The first method adjusts the dissimilarity function by using weighting functions while the second method redefines the features space by means of linear and nonlinear transformation functions. Experimental results on real data sets demonstrate that our methods are effective and the results show that the transformation approach outperforms the weighting approach, achieving a precision gain of up to 70%. Our results indicate that nonlinear transformations have a great potential in capturing the user's interests in image retrieval and should be further analyzed employing other learning/optimization mechanisms[1].

Categories and Subject Descriptors

H.3.3 [**Information Search and Retrieval**]: Search Process, Relevance Feedback.

General Terms

Algorithm, Experimentation.

Keywords

Image Retrieval, Genetic Algorithm, Weighting, Functions.

1. INTRODUCTION

Techniques for image retrieval follow two main approaches [8]: Text-Based Image Retrieval (TBIR) and Content-Based Image Retrieval (CBIR). TBIR techniques uses descriptions provided by textual annotation, which may introduce inconsistencies due to the human annotator. This is due to the fact that, in many domains, text cannot accurately capture the visual attributes of an image based on human perception. CBIR techniques, in turn, use content-based description instead of textual description. In CBIR, the images are indexed/retrieved considering their extracted visual content, such as color, texture and shape features [1]. Such features together define *feature vectors* containing m elements that are interpreted as points in an m-dimensional space. In the features space, one assumes that a query point is surrounded by points that represent the most similar images to a given image of interest, an operation well-known as similarity query. Such query are appropriately calculated with the application of dissimilarity functions, one of the basis of CBIR.

Despite dealing with inherent information obtained from the images, CBIR systems often present inaccurate results due to the challenging problem of associating low-level features with the high-level semantics of the images. This lack of correspondence between the high-level similarity from the point of view of the user and the low-level image features is known as *semantic gap* [17]. This problem can be caused, for example, by assuming that all features are equally relevant no matter the objective of the image retrieval. In this sense, some features can be very representative for some queries while being irrelevant for other queries. Also, in a given CBIR context, some features have poor or no semantic meaning, while other features are successful in capturing the semantics.

As an attempt to attenuate the semantic gap problem, Relevance Feedback (RF) methods have been proposed [3] [6] [20]. RF methods are very suited to the task of providing to a CBIR system a mechanism that allows it to learn which features best capture the user's interests.

In the RF process, the user is supposed to evaluate the images retrieved in the current query by assigning them values that state their relevance, semantically speaking. After this step the system reformulates the preceding query, taking into account the user's evaluations to improve its results. In many cases the relevant feedback problem itself is handled as a search problem related to weights, parameters, and/or data aggregation models, such as functions combining multiple descriptors. A review on RF for image retrieval is presented in [23].

In regarding search problems, Genetic Algorithms (GAs) provide a general adaptive search methodology based on natural selection; a methodology that has been successfully employed to perform feature selection and weighting on dissimilarity functions used in CBIR systems. In the realm of

[1]This work is based on an earlier work: SAC'12 Proceedings of the 2012 ACM Symposium on Applied Computing, Copyright 2012 ACM 978-1-4503-0857-1/12/03. http://doi.acm.org/10.1145/2245276.2245471.

CBIR systems tuned by Relevance Feedback techniques, this study proposes two GA-based RF methods to enhance the accuracy of image retrieval tasks:

- the first method adjusts the dissimilarity calculus by means of weighting functions that calibrate the importance and impact of each feature in a given features space;
- the second method transforms the features space through linear and non linear transformation functions.

The remainder of this paper is structured as follows. Section 2 presents the related work. Section 3 presents the preliminaries and notations that we use throughout the paper. Section 4 formally describes the proposed methods. Section 5 details the experimental evaluation. Finally, Section 6 presents the conclusions and points out future works.

2. RELATED WORK

Researches on improving image retrieval effectiveness mainly employ RF [23], dissimilarity function learning [21] and feature selection [15] methods. The most used RF approach employs dynamic weighting mechanisms to modify the distance function or image dissimilarity model through appropriate weights, so that the distance between relevant images becomes smaller if compared to the non-relevant images.

In the study of Stejic *et al.* [18], the authors incorporate GA into RF mechanisms in order to assign the appropriate weights on image descriptors and image regions. However, the authors did not provide an effective model to learn the user's requests, because the *R-precision* evaluation function that they employed represents only the ratio of retrieved relevant images. Differently, our study addresses this question through a GA-based RF mechanism that relies on an order-based ranking evaluation function.

Based on the premise that dissimilarity functions and image descriptors are problem-oriented, Torres *et al.* [21] proposed the use of nonlinear combinations of multiple image similarities, addressing the problem through the use of Genetic Programming (GP). In their investigation, the learning process relies on a training set and not on the user's feedback. Thus, the outcomes of their methods are not adjusted to the user's transient interests.

Other studies attempt to improve the precision of CBIR systems by working directly with the features space [13] [5]. The study of Silva *et al.* [5] relies on ranking evaluation functions in order to choose the best set of features to represent images in the CBIR context; the contribution of each feature is binary (selected or not selected). In a different course of action, our methods take advantage of the relative importance of each feature in image retrieval, considerably improving the retrieval results.

3. PRELIMINARIES AND NOTATION

Let $\mathbf{X} = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$ represent the set of feature vectors extracted from the image database $\mathbf{I} = \{\mathbf{i}_1, \ldots, \mathbf{i}_n\}$ using the feature extractor ε, i.e., $\varepsilon(\mathbf{i}_i) = \mathbf{x}_i = \{x_1, \ldots, x_m\}$, $\mathbf{x}_i \in \mathbb{R}^m$. Now consider $\mathbf{x}_q$ as being the feature vector extracted from the query image $\mathbf{i}_q$ and $\mathbf{x}_i$ the feature vector extracted from an arbitrary image $\mathbf{i}_i$. A dissimilarity function $d()$, also called distance function, provides a mechanism to measure the dissimilarity between $\mathbf{x}_q$ and $\mathbf{x}_i$.

Since the smaller the dissimilarity the larger the similarity is, the elements of $\mathbf{X}$ can be sorted according to their similarities to $\mathbf{x}_q$. In CBIR the most popular form of similarity query is the k-Nearest Neighbor query, or kNN query for short.

The kNN query aims at finding the k closest objects to a query object [22]. A kNN query is formally presented in Definition 1.

Definition 1. Given the set of objects $\mathbf{X}$, a distance function $d()$ and the query object $\mathbf{x}_q$, the response set $\mathbf{T}$ of a kNN query is defined as:

$$kNN(\mathbf{x}_q, k, d(), \mathbf{X}) = \{\mathbf{T} \subseteq \mathbf{X}, |\mathbf{T}| = k \wedge \forall \mathbf{x}_i \in \mathbf{T}, \mathbf{x}_j \in \mathbf{X} \setminus \mathbf{T} : d(\mathbf{x}_q, \mathbf{x}_i) \leq d(\mathbf{x}_q, \mathbf{x}_j)\} \quad (1)$$

We consider that the elements of $\mathbf{T}$ are sorted according to their distances to the query image composing a ranking $\mathcal{T} = (\mathbf{t}_1 \prec \mathbf{t}_2 \prec \ldots \prec \mathbf{t}_{k-1} \prec \mathbf{t}_k)$, where $\forall i = \{2, \ldots, k\}$, $d(\mathbf{x}_q, \mathbf{t}_{i-1}) \leq d(\mathbf{x}_q, \mathbf{t}_i)$.

In order to evaluate the effectiveness of kNN queries into the optimization process of RF, we employed a ranking quality measure, as described in Definition 2. This is an order-based utility function that considers the utility of images retrieved according to their ranks [11]. Relevant images in the first positions of the ranking will receive higher scores of utility while relevant images far from the ranking top will receive lower scores.

Definition 2. Given the ranking $\mathcal{T}_i = (\mathbf{t}_1, \ldots, \mathbf{t}_k)$ as the result of the query $kNN_i(\mathbf{x}_q, k, d(), \mathbf{X})$ and $\mathbf{R}_q = \{\mathbf{r}_1, \ldots, \mathbf{r}_\rho\}$, $\mathbf{R}_q \subseteq \mathbf{X}$ the set of objects that belong to the same class of $\mathbf{x}_q$, also called here the relevant objects for $\mathbf{x}_q$, the measure of the quality of $\mathcal{T}_i$ is calculated by the function:

$$\Phi(\mathcal{T}_i, \mathbf{R}_q) = \sum_{j=1}^{k} \frac{r(\mathbf{t}_j)}{A} \cdot \left(\frac{(A-1)}{A}\right)^{j-1} \quad (2)$$

where $r(\mathbf{t}_j) = 1$, if $\mathbf{t}_j \in \mathbf{R}_q$ or $r(\mathbf{t}_j) = 0$ otherwise, and $A \geq 2$ is an adjustment parameter that expresses the relative importance of the position of the elements on the ranking. Small values for A means more importance for the relevant elements on the first positions of the ranking. When A is large, the fraction $\frac{(A-1)}{A}$ is close to 1, then the position of the elements on the ranking is not strongly considered.

We apply the proposed ranking-quality measure as the fitness function at the core of the Genetic Algorithm used along this research. Our goal relies on achieving a ranking that maximizes the value of this function.

4. PROPOSED METHODS

4.1 Overview of the System

The scheme of the system is shown in Figure 1. We suppose that we have an image data set $\mathbf{I}$ with the image features extracted using a feature extractor ε. Initially, the user enters a query image $\mathbf{i}_q$ and we apply ε on $\mathbf{i}_q$ to get the feature vector $\mathbf{x}_q$, i.e. $\mathbf{x}_q = \varepsilon(\mathbf{i}_q)$. Then, the system computes the distance between the respective feature vector $\mathbf{x}_q$ to the features of the images from data set $\mathbf{I}$.

After this, a ranking is generated and presented to the user, who is in charge of evaluating the first k images in the ranking, assigning them relevance values: *relevant*, $r(\mathbf{t}_j) = 1$ if the image is relevant to the query; and *not desirable*, $r(\mathbf{t}_j) = -1$ if the image is not supposed to be in the ranking. By default, the system assigns *not relevant*, $r(\mathbf{t}_j) = 0$ to every image that eventually has not been evaluated by the user. This iterative process defines the set of relevant objects $\mathbf{R}_q$. Since many images can belong to multiple classes even according to the user's judgment, it is the specific user's need that will define whenever an image is related or not to the image query $\mathbf{i}_q$.

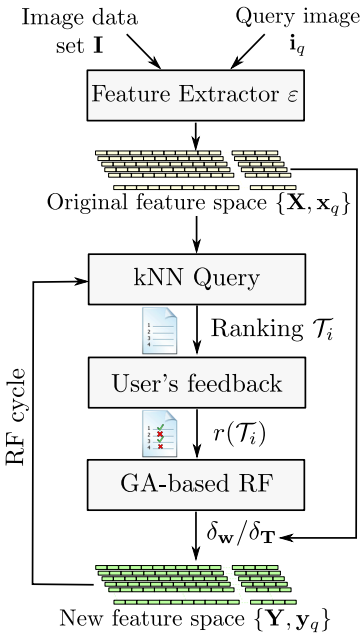


Figure 1: Systematization of the proposed method.

Once the user's feedback is provided, we apply a GA-based search for a sequence of *weighting functions* or *transformation functions* ($\delta_{\mathbf{w}}$ or $\delta_{\mathbf{T}}$) that that will maximize the fitness function presented in Equation 2. The application of δ_w and δ_T to adjust the content-based retrieval are presented formally in subsections 4.3 and 4.4, respectively. In summary, a sequence of weighting functions is inferred in order to adjust the distance function so to reflect the user feedback; and a sequence of transformation functions is determined in order to transform the original feature space $\mathbf{X}$ and the feature vector of the query $\mathbf{x}_q$ so to achieve more accuracy in retrieval tasks.

Values of relevance provided by the user are stored in a temporary space in between successive RF iterations. The cycles of RF/GA-search are repeated until the optimal solution is found, which means that the resulting ranking contains, as much as possible, only relevant images in its first k positions; or until a predefined maximum number of cycles is reached.

In short, the approaches we propose are:

- inferring a weight vector by means of weighting functions; this allows the distance function to take into account the degree of contribution of each feature according to the user's feedback;
- optimizing the features space by inferring a space transformation that will adjust the original space in order to better represent the user's similarity criteria.

4.2 Weighting Functions Approach

The use of GA to infer the most appropriate weight vectors for the dissimilarity function has achieved promising results as pointed out in previous studies. Accordingly, our first approach is based on this same concept but, in an innovative fashion, we generate weights that obey to well-known mathematical functions, as opposed to former methods that use constant values. This proposal came from investigating the distribution of the dissimilarity distances before and after the use of a weight vector. The assumption was that there could be a well-defined manner to determine the best weight to each feature so to improve the ranking generated by the similarity queries. In accordance, we implemented a CBIR system that integrates Relevance Feedback and Genetic Algorithms in order to find weight vectors in the form of sequences of mathematical functions. For this intent, we considered a vast set of functions, as illustrated in Figure 2.

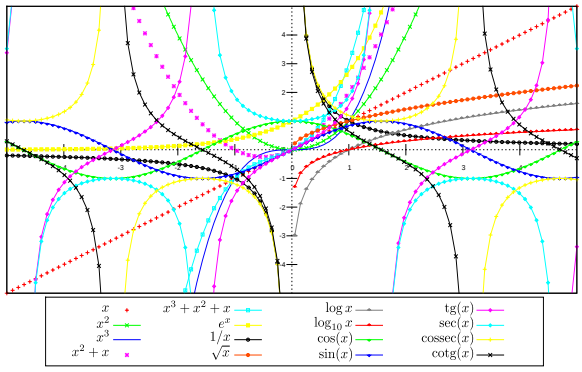


Figure 2: Examples of non linear and linear functions used to generate the inference of weights and features transformations.

The problem, formally presented in Definition 3, consists in finding a sequence of functions that generate the best weight vector for a specific query. In our problem, the chromosome used by the GA search is encoded slightly different from the most usual approach, according to which real values corresponding to the weights are assigned to genotype. In our case, the values assigned to the genotype are integers that correspond to the identifiers of the weighting functions.

Definition 3. Let $\mathbf{F} = \{f_1, \ldots, f_\rho\}$ be a set of mathematical functions and $\delta_\mathbf{w} = (f'_1, \ldots, f'_m)$, $f'_i \in \mathbf{F}$ be the ordered set of weighting functions inferred. The number of elements of the sequence δ_w must be equal to the number of features employed for images descriptions. Now let $\mathbf{w} = \delta_\mathbf{w}(\mathbf{x}_q) = \{f'_1(x_{q_1}), \ldots, f'_m(x_{q_m})\}$ be the weighting vector for the distance function $d_\mathbf{w}()$, where $f'_i(\mathbf{x}_{qi})$ is a transformation applied on the i-th feature of query image ($\mathbf{x}_{qi}$) to produce a weight for the i^{th} feature in the dissimilarity function. The problem using the weighting functions is to find a set $\delta_\mathbf{w}$ such that

$$\arg\max_{\delta_\mathbf{w}} \Phi(\mathcal{T}, \mathbf{R}_q) \tag{3}$$

where $\mathcal{T} = kNN(\mathbf{x}_q, k, d_\mathbf{w}(), \mathbf{X})$ and $\mathbf{R}_q$ is the relevance values for each image regarding the query represented by feature vector x_q.

The advantage of dealing with weighting functions is that the search space for the GA is significantly reduced in comparison with the usual weighting over a given interval, such as $[0, 1]$. The search space in a continuous interval is much higher than the discrete and finite search space $\mathbf{F}$, according to, for each weight, we have only $|\mathbf{F}|$ possible choices.

4.3 Feature Space Transformation Functions Approach

The second approach we introduce is based on the transformation of the features space, according to which each feature value is redefined by a transformation function, as detailed in Definition 4. These functions provide linear and nonlinear transformations over the original features space, a way to capture nonlinear relationships.

Transformations over the features space are defined using the GA search to find the sequence of functions that lead to improved retrieval results. For each sequence of transformation functions considered by the GA, a new feature space is calculated using the original feature values; after each consideration, a new kNN query is executed.

Definition 4. Given $\mathbf{F}$, as previously set out, let $\delta_\mathbf{T}$, defined analogously to $\delta_\mathbf{w}$, be an ordered set of inferred transformation functions and $\mathbf{Y} = \{\mathbf{y}_1, \ldots, \mathbf{y}_n\}$ the new feature space, and each $\mathbf{y}_i$ corresponding to a $\mathbf{x}_i$ transformed by the functions in $\delta_\mathbf{T}$, i.e., $\mathbf{y}_i = \delta_\mathbf{T}(\mathbf{x}_i) = \{f'_1(x_{i_1}), \ldots, f'_m(x_{i_m})\}$.

The problem translates into finding the set $\delta_\mathbf{T}$ of transformation functions such that:

$$\arg\max_{\delta_\mathbf{T}} \Phi(\mathcal{T}, \mathbf{R}_q) \tag{4}$$

where $\mathcal{T} = kNN(\mathbf{y}_q, k, d(), \mathbf{Y})$.

4.4 Genetic Algorithm Description

When implementing a GA, it is necessary to consider its parameters and operators, such as the chromosome coding, the fitness function, selection, the crossover, and the mutation operators. The values of the parameters and the chosen operators can be decisive regarding the effectiveness of the algorithm. In this investigation, such choices were made experimentally and are described as follows:

- *Chromosome coding*: for the two proposed methods, a chromosome was coded as an integer-valued vector with m positions, $\mathbf{C} = (g_1, \ldots, g_m)$, where g_i corresponds to an identifier of a function in $\mathbf{F}$. In the weighting functions approach, the chromosome produces the weight vector $\mathbf{w}$ for the distance function $d_\mathbf{w}()$; while in the features space transformation approach, it provides the new features space $\mathbf{Y}$.
- *Fitness function*: as fitness function we employed the ranking quality measure presented in Definition 2.
- *Selection for recombination operator*: we used exponential ranking selection to select pairs of individuals to reproduce. For an individual $\mathbf{C}_i$, the probability p_i of being selected is given by Equation 5:

$$p_i = \frac{c^{Sp-i}}{\sum_{j=1}^{Sp} c^{Sp-j}}, 0 \leq c \leq 1 \tag{5}$$

  where $i \in \{1, \ldots, Sp\}$, Sp is the population size and $c = 0.9$.
- *Selection for reinsertion operator*: elitism was employed to select the surviving chromosome for the next generation. This is because elitism guarantees that the best individual of the population in a generation g will be present in the population of generation $g+1$, and so it guarantees that the best solution will be improved or, at least, maintained.
- *Crossover operator*: we employed uniform crossover. A mask is randomly built and it indicates which chromosome will supply each gene for the first offspring. The second offspring is generated by the complement of the same mask.
- *Mutation operator*: uniform mutation is applied on the offspring chromosomes. The genes are selected with probability P_m and their values are changed by another valid value randomly chosen.

5. EXPERIMENTAL RESULTS AND ANALYSIS

We evaluated our proposed methods by comparing it with the weighting approach most commonly used, according to which the weights correspond to values in the range $[0, 1]$ – here, we named this approach *direct weights generator (WG)*. This section describes the setting of the experiments and the analysis of the results.

5.1 Image Data sets

In order to assess the applicability of the proposed methods, experiments were conducted on real data sets in two different domains: medical and general domain.

General Domain

The first image database from a general domain is *Corel 1000* [9]. This data set contains 1000 images classified into 10 classes with 100 images per class (*africa*, *beach*, *buildings*, *buses*, *dinosaurs*, *elephants*, *flowers*, *food*, *horses* and *mountains*).

The other general domain data set used for experiments is *Scenes* [14]. This data set is comprised of 1678 images divided into 5 classes: 2 classes of urban environments (*highway* with 260 images and *tall buildings* with 356 images), and 3 classes of natural environments (*coast* with 360 images, *forest* with 328 images, and *mountain* with 374 images).

Medical Domain

From the medical domain we used two data sets, one related to general exams and one more specific. These data sets were collected at the Clinical Hospital at University of São Paulo in Ribeirão Preto.

The first data set called *Medical Exams* is a collection of 2400 medical images of various body parts. The images were extracted by X-Ray and MRI exams, classified according to the body part and cut type. The base is split into 12 classes (*abdomen*, *axial brain*, *coronal brain*, *sagittal brain*, *breast*, *chest*, *leg*, *hand*, *knee*, *lung*, *pelvis* and *spine sagittal*), each class contain 200 images.

The second data set, called *Lung*, consists of 246 MRI exams of human lungs. This data set is divided into 6 classes being 1 of normal and 5 of abnormal patterns (*consolidation*, *emphysema*, *thickness*, *honeycombing*, and *ground-glass opacity*), with an average of 41 images per class, varying from 39 to 44.

5.2 Parameters Setup

Feature Extractors

For each data set, the feature vectors were acquired by the feature extractors: *Color Moments* [19], *Co-occurrence* [7], *Sobel Histogram* [2], *Histogram* [4], *Run Length* [10] and *SIFT* [12]. Table 1 shows the number of features extracted and the type of the information captured by each extractor. When extracting *Color Moments* and *Sobel Histogram*, the images were partitioned into 16 rectangular regions. The respective features were extracted from each region and combined in a single vector.

Table 1: Feature extractors employed

Feature extractor	Number of features	Type
Color Moments	144	Color
Co-occurrence	88	Texture
Sobel Histogram	128	Shape
Histogram	256	Color
Run Length	44	Texture
SIFT	128	Shape

All the features were normalized using the *z-score* function to avoid *bias* on distance calculation. The dissimilarity measures were obtained by the Euclidean distance function (L_2). The Weighted Euclidean distance function was applied to the weighting methods.

One important issue concerned to the feature extractors is that if the extracted features do not describe the relevant aspect of the user's interests, the GA may not converge to satisfactory solutions. This is due to the fact that the set $\mathbf{R}_q$ will be empty and therefore won't be able to contribute to the fitness computation.

GA Parameters

After some previous experiments and analysis, it was observed that the values assigned to the GA parameters that achieved better results were:

- population size (Sp): 50;
- maximum number of generations (Ng): 100;
- crossover rate (Pc): 0.8;
- mutation rate (Pm): 0.02.

5.3 Results

All the experiments were performed using the three following methods: direct weights generator (WG), weighting functions (WF) and transformation functions (TF). All the methods (WG, WF e TF) employ the fitness function of Equation 2, over which we test the values 2, 10 and 20 for parameter A. Also, we analyze the performance of the different feature extractors. Regarding the kNN query, we empirically selected $k = 30$ and the maximum number of cycles of RF/GA-search is 10.

The effectiveness of our methods WF and TF, in comparison with the WG method, was assessed by method Precision *vs.* Recall (P&R), by method Number of Relevant Images *vs.* Cycles of RF, and by visual data analysis. The qualitative visual analysis aimed at verifying the distribution of the relevant and non relevant images in the resulting optimized spaces, and aimed at measuring the cohesion of the cluster composed of relevant images.

The visual data analysis employed here was obtained with tool *Metric Space Platform* (*MetricSplat*) [16]. This tool combines the benefits of visualization techniques with methodologies for content-based image retrieval. Fastmap projection (cloud of point) was the technique chosen to illustrate features space configuration before and after optimizations.

In the visualization process, for each data set one image was randomly chosen. Then, we compared the results of the initial query and the results given after applying the optimization methods WF and TF. In order to allow a better visual analysis, for each visualization, we considered the entire data space and, comparatively, a reduced space with only the first 50 elements. Red dots represent the relevant elements of the query and the blue dots represent elements that are not relevant.

The cohesion measure was used to quantify how close are the images that belong to the same class of the query in relation to the query center. The cohesion measures were taken before and after the optimizations. We used measure Mean Square Deviation (MSD), calculated as follows:

$$MSD(C) = \frac{1}{n}\sum_{i=1}^{k}(c - x_i)^2 \qquad (6)$$

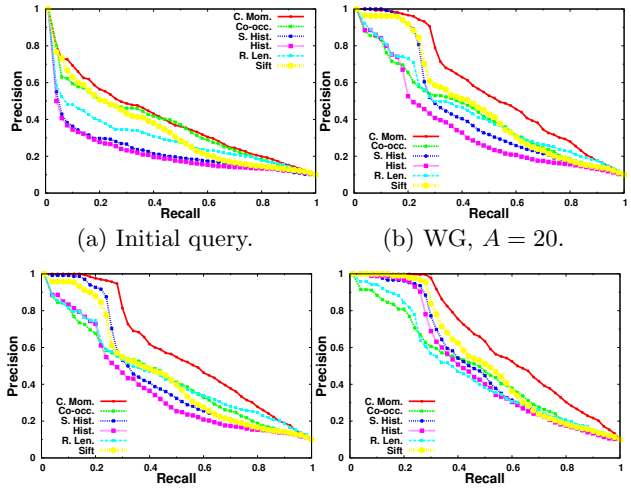
where c is the centroid (the query in this case) and x_i is an element of cluster C (here it is considering only the cluster composed of relevant elements). Small values for MSD indicate better cohesion on clusters.

5.3.1 Corel 1000

Experiments on *Corel 1000* are presented considering an average of 10 image queries randomly chosen, one from each class. Figure 3(a) shows low precision of each feature extractor in the initial query and the improvement on precision obtained in the last RF cycle using each method (Figures 3(b), (c) and (d)). It can be observed that the WF (Figure 3(c)) obtains higher precision then WG (Figure 3(b)), near to 20% of recall with *Color Moments* feature extractor; TF (Figure 3(d)), in turn, achieved the highest precision, near to 30% of recall, also using *Color Moments*.

Considering extractor *Color Moments*, which was the best extractor for the *Corel 1000* data set, in Figure 4(a) we analyze the adjustment of the parameter A of the fitness function for each method. Figure 4(a) shows that the best results were achieved when using TF, with $A = 20$, $A = 10$ and $A = 2$, respectively. Following, one can see the WF method with $A = 20$ and $A = 10$. In Figure 4(b) it is possible to observe that the precision increases until the fifth cycle, with low or no improvement after that. We believe that the Color Moments were the best extractor for the Corel 1000 data set because it comprises image classes each with a characteristic color hue.

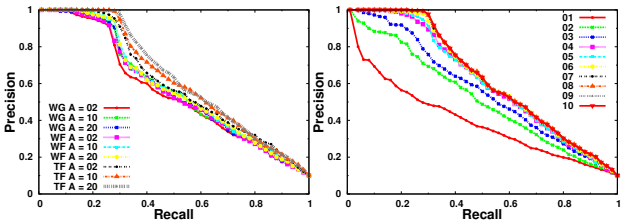
The number of relevant images retrieved per cycle obtained by each feature extractor using TF method with $A = 20$ is shown on Figure 5(a). It can be observed that the texture-based feature extractors, *Co-occurrence* and *Run Length*, have similar and poor improvement while compared to the other ones. Figure 5(b) shows the number of relevant images retrieved through the cycles for each method with $A = 2, 10, 20$ and using the *Color Moment* extractor. As we can see, the TF method was the most effective, followed by the WF method for the average of all feature extractors.



(a) Initial query. (b) WG, $A = 20$.

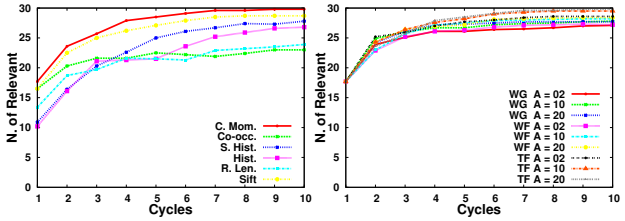
(c) WF, $A = 20$. (d) TF, $A = 20$.

Figure 3: P&R plot for each feature extractor and $A = 20$ (a) initial query, (b) cycle 10 using WG, (c) cycle 10 using WF, (d) cycle 10 using TF.



(a) Cycle 10. (b) TF, $A = 20$.

Figure 4: P&R plot for the feature extractor *Color Moments* (a) cycle 10 for each method and for each value of A, (b) evolution through the cycles 01 to 10 using TF and $A = 20$.



(a) TF, $A = 20$. (b) *Color Moments*.

Figure 5: Number of Relevant (a) for each feature extractor using TF and $A = 20$, (b) using *Color Moments* for each method and for each value of A.

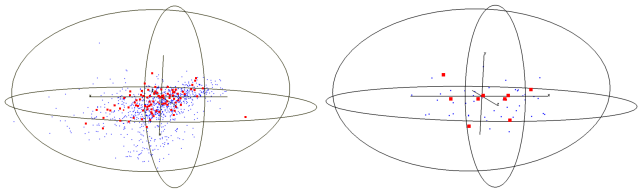
Figure 6 shows the projection of the original space from *Corel 1000* using *Color Moments*. The configuration of the spaces after using methods WG and WF are respectively illustrated in Figure 7 and Figure 8. Figures 7(a) and 8(a) show that the generated spaces preserve a sparse distribution of the elements, similar to the original space (Figure 6(a)). WG retrieved 16 relevant elements (Figure 7(b)) while WF retrieved 17 (Figure 7(b)) among the first 50 elements retrieved. Notice that, in the visualizations, the space scales are not fixed.

Comparing the TF method with the weighting approaches, it can be seen (Figure 9(a)) that the generated space is more compact, and the relevant elements are concentrated closer to the query center. Figure 9(b) shows that TF retrieved 33 relevant images of the 50 first elements in the ranking; 94% more accurate than WG and WF.

The bar chart in Figure 10 shows the cohesion for each space configuration. The cluster of relevant images obtained by the initial query had the higher cohesion; meanwhile, the cluster obtained by TF presented the best value. Considering the weighting methods, WF was superior than WG on its space configuration.

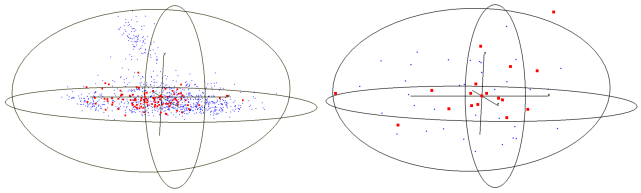
5.3.2 Scenes

Figure 11(a) illustrates the precision *vs.* recall results for the initial queries on data set *Scenes*. It can be seen that the *Co-occurrence* extractor achieved the best result while *Histogram* achieved the worst result. After 10 optimization cycles, it can be observed in Figures 11(b), (c) and (d) that the extractor *Sobel Histogram* achieved the best results for all methods. WG was more accurate when assigned $A = 20$, while WF and TF had better results with $A = 10$. Fixing recall at 20%, the average precision obtained by TF was 67%,



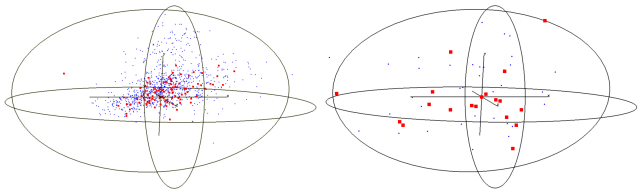
(a) (b)

Figure 6: Original space configuration of *Corel 1000* using *Color Moments* (a) entire space (b) 50 nearest elements from the query point.



(a) (b)

Figure 7: Data space configuration of *Corel 1000* data set after applying WG method (a) entire space (b) 50 nearest elements from the query point.



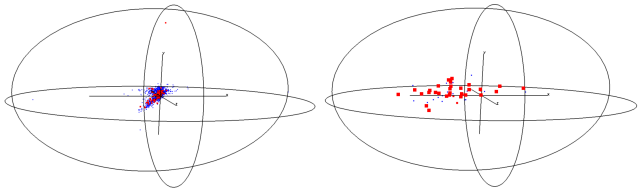
(a) (b)

Figure 8: Data space configuration of *Corel 1000* data set after applying WF method (a) entire space (b) 50 nearest elements from the query point.

superior to WF with 64% and WG with 61%. The Sobel Histogram achieved the best results because this descriptor can capture the complex patterns present in images with a high number of edges, which is the case for the Scenes data set.

Regarding extractor *Sobel Histogram*, Figure 12(a) shows the results of the last cycle for each method and the values assigned to parameter A. The maximum precision was achieved by methods TF ($A = 2$, 10 and 20) and WF ($A = 10$) up to the recall rate of 8%. As Figure 12(b) shows, the improvements for method TF with $A = 10$ were more significant until the fourth cycle, that is, the method converged at this cycle.

The graphics in Figure 13 present the number of relevant images retrieved per cycle. Figure 13(a) shows results obtained by TF with $A = 10$, where the more effective extractor was the *Sobel Histogram*, followed by *SIFT* on the last cycles. For extractor *Sobel Histogram*, the methods TF and WF (as illustrated on Figure 13(b)) were slightly more effective while using $A = 10$, in contrast to WG, which was more effective with $A = 20$. TF achieved an average of 30 relevant retrieved images, while WF and WG achieved 29 and 28,



(a) (b)

Figure 9: Data space configuration of *Corel 1000* data set after applying TF method (a) entire space (b) 50 nearest elements from the query point.

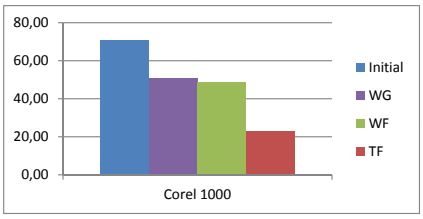
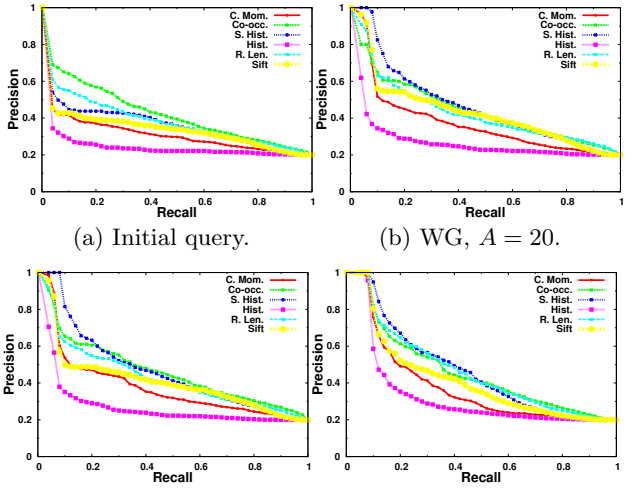


Figure 10: MSD values for relevant elements clusters on *Corel 1000* data set. The smaller the MSD value the higher cohesion.



(a) Initial query. (b) WG, $A = 20$.

(c) WF, $A = 10$. (d) TF, $A = 10$.

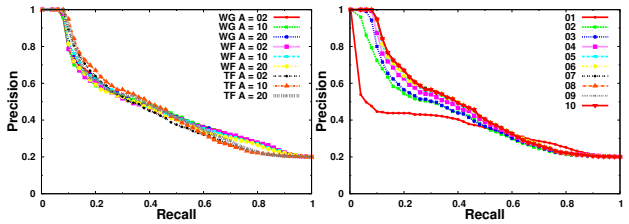
Figure 11: P&R plot for each feature extractor (a) initial query, (b) cycle 10 using WG, $A = 20$, (c) cycle 10 using WF, $A = 10$, (d) cycle 10 using TF, $A = 10$.

respectively.

Figure 14 shows the visualization of the original data space using *Sobel Histogram*. The space obtained by TF (Figure 17(a)) was better adjusted to the query because the relevant elements were closer to the center, different from the spaces obtained by WG and WF (Figures 15(a) and 16(a)).

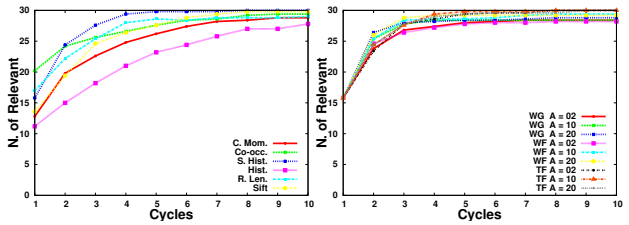
All optimization methods increased the number of relevant images considering the 50 elements closer to the center, as Figures 15(b), 16(b) and 17(b) show. The best results were obtained by WF and TF, both with 36 relevant elements in the first 50 elements, while WG achieved 21 relevant elements. Figure 18 shows the MSD values for the cluster with relevant elements. TF has a slightly better result if

compared to WF and to WG; all the three methods had a significantly smaller MSD for the relevant cluster, which means that cohesion increased and so the optimizations were very effective.



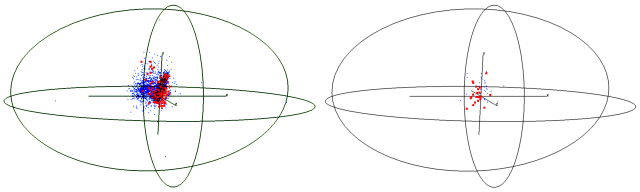
(a) Cycle 10. (b) TF, $A = 10$.

Figure 12: P&R plot for the feature extractor *Sobel Histogram* (a) cycle 10 for each method and for each value of A, (b) evolution through the cycles 01 to 10 using TF and $A = 10$.



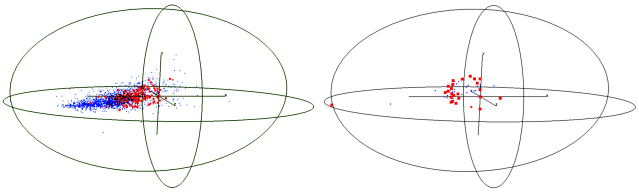
(a) TF, $A = 10$. (b) *Sobel Histogram.*

Figure 13: Number of Relevant (a) for each feature extractor using TF and $A = 10$, (b) using *Sobel Histogram* for each method and for each value of A.



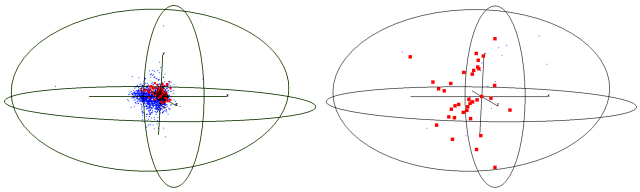
(a) (b)

Figure 14: Original space configuration of *Scenes* using *Sobel Histogram* (a) entire space (b) 50 nearest elements from the query point.



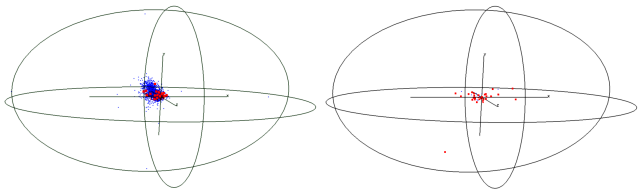
(a) (b)

Figure 15: Data space configuration of *Scenes* data set after applying WG method (a) entire space (b) 50 nearest elements from the query point.



(a) (b)

Figure 16: Data space configuration of *Scenes* data set after applying WF method (a) entire space (b) 50 nearest elements from the query point.



(a) (b)

Figure 17: Data space configuration of *Scenes* data set after applying TF method (a) entire space (b) 50 nearest elements from the query point.

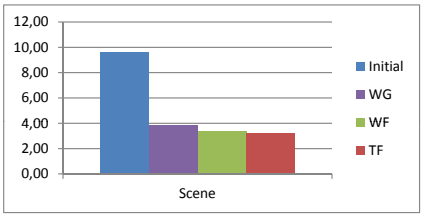


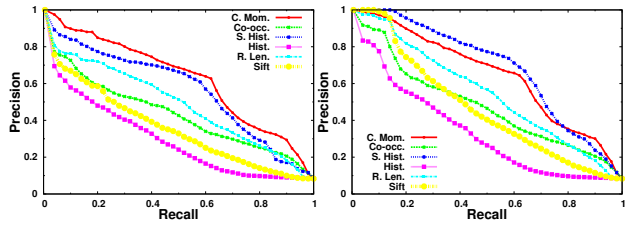
Figure 18: MSD values for relevant elements clusters on *Scenes* data set.

5.3.3 *Medical Images*

The average P&R results for the initial queries on the *Medical Images* data set are shown in Figure 19(a). *Color Moments* presented the best initial results, with an average of 26 relevant images retrieved, whilst *Histogram* obtained the worst initial results, as it retrieved an average of 17 relevant images considering the first 30 images retrieved. For this data set, extractor *Sobel Histogram* was more accurate after the optimization using WG ($A = 20$), WF ($A = 10$) and TF ($A = 10$) methods, as can be seen in Figures 19(b), (c) and (d), respectively). Considering recall rate at 20%, the average precision was near 95% for all methods. Texture-based extractors achieved the best results after optimization using TF; this result is expected as medical images are highly characterized by textural content.

All three methods achieved maximum precision while using *Sobel Histogram* up to 14% of recall, as it is shown in Figure 20(a). Improvements on accuracy through the cycles while using TF and $A = 10$ are shown in Figure 20(b). Once more, the convergence occurs around the fourth cycle.

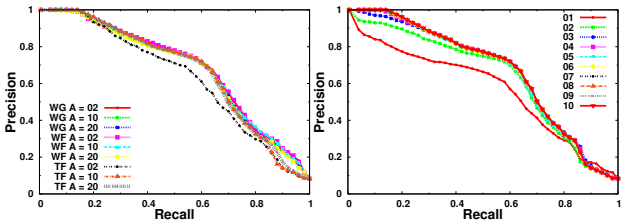
The plots of the number of relevant results per cycle in Figure 21 show that extractor *Sobel Histogram* had the best response regarding TF optimization, followed by *SIFT*(Figure 21(a)). Figure 21(b) also shows that all meth-



(a) Initial query. (b) WG, $A = 20$.

(c) WF, $A = 10$. (d) TF, $A = 10$.

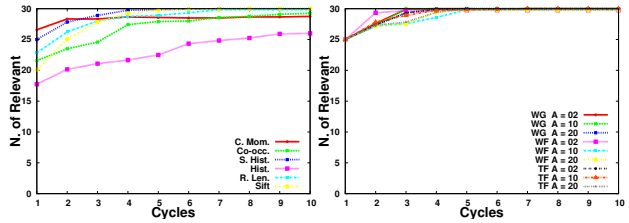
Figure 19: P&R plot for each feature extractor and (a) initial query, (b) cycle 10 using WG, $A = 20$, (c) cycle 10 using WF, $A = 10$, (d) cycle 10 using TF, $A = 10$.



(a) Cycle 10. (b) TF, $A = 10$.

Figure 20: P&R plot for the feature extractor *Sobel Histogram* (a) cycle 10 for each method and for each value of A, (b) evolution through the cycles 01 to 10 using TF and $A = 10$.

ods achieved 30 relevant images considering the 30 first images retrieved in the fifth cycle while optimizing features extracted by *Sobel Histogram*.

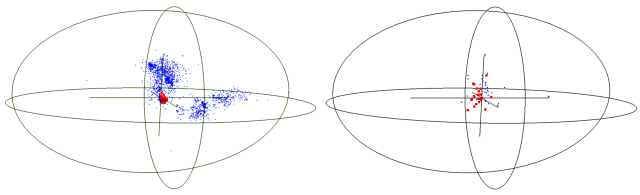


(a) TF, $A = 10$. (b) *Sobel Histogram*.

Figure 21: Number of Relevant (a) for each feature extractor using TF and $A = 10$, (b) using *Sobel Histogram* for each method and for each value of A.

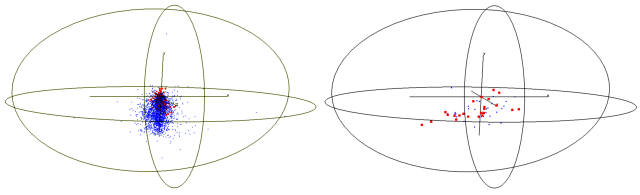
The visualizations of the *Medical Images* data set correspond to the spaces generated by the features extracted using *Sobel Histogram*. Figure 22 illustrates the initial configuration of this space. It is noticeable that there is a concentration of relevant images near to the center of the space (Figure 22(a)).

The distribution of the elements on space generated by WG (Figure 23(a)) reveals that the method brought the non relevant images closer to the center in comparison to the initial space. The methods WF (Figure 24(a)) and TF (Figure 25(b)) generated better configurations. Both methods preserved a separation of non relevant elements. TF was more effective since the cluster of the relevant images was very close to the center. Considering the 50 elements closer to the center, WF retrieved 34 relevant ones (Figure 24(b)) and TF (Figure 25(b)) retrieved 36, while WG retrieved 25 (Figure 23(b)). Measures of cohesion on the relevant clusters confirm the visualizations. As shown in Figure 26, TF obtained the lowest value, with significant difference from the values on WG and WF.



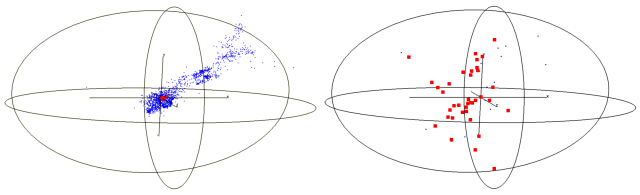
(a) (b)

Figure 22: Original space configuration of *Medical Images* using *Sobel Histogram* (a) entire space (b) 50 nearest elements from the query point.



(a) (b)

Figure 23: Data space configuration of *Medical Images* data set after applying WG method (a) entire space (b) 50 nearest elements from the query point.

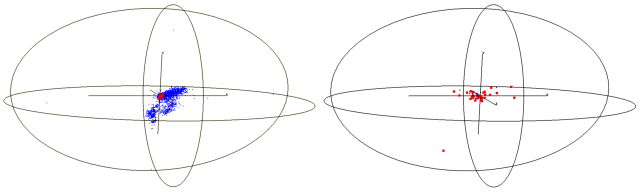


(a) (b)

Figure 24: Data space configuration of *Medical Images* data set after applying WF method (a) entire space (b) 50 nearest elements from the query point.

5.3.4 *Lung*

For this data set we have randomly chosen 6 images, one from each class; following we present and the average results obtained with the respective queries. Figure 27(a) shows the low precision of each feature extractor of the initial query, with the best performance, again, obtained by using *Color Moments*. Figure 27(b) and 27(c) illustrate the



(a) (b)

Figure 25: Data space configuration of *Medical Images* data set after applying TF method (a) entire space (b) 50 nearest elements from the query point.

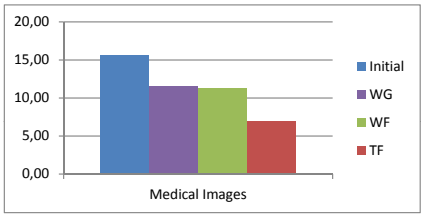
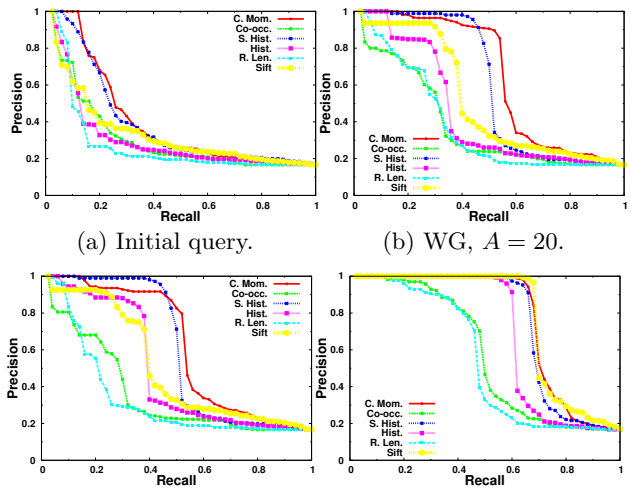


Figure 26: MSD values of clusters of relevant elements on *Medical Images* data set.

performance of WG and WF methods respectively, with a slightly better effectiveness of WF and best improvement achieved by the feature vector *Sobel Histogram* in both cases. Figure 27(d) shows significantly higher effectiveness of TF method (with 100% of precision until 60% of recall) in comparison to both WG and WF, and the best improvement achieved using *SIFT*.

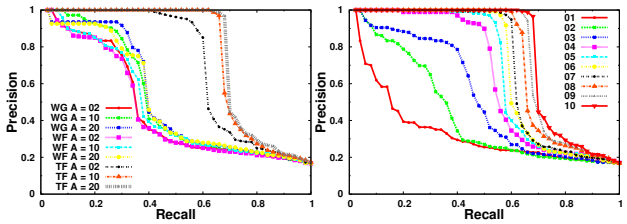


(a) Initial query. (b) WG, $A = 20$.

(c) WF, $A = 20$. (d) TF, $A = 20$.

Figure 27: P&R graphics for each feature extractor and $A = 20$ (a) initial query, (b) cycle 10 using WG, (c) cycle 10 using WF, (d) cycle 10 using TF.

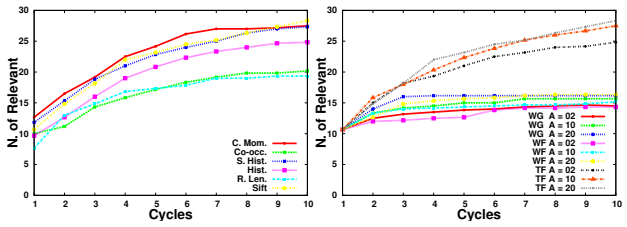
Considering the feature extractor *SIFT*, Figure 28(a) shows the highest precision when using TF and $A = 20$, 10 and 2, in decreasing order. Furthermore, the performance difference in using weighting and transformation techniques is noticeable. For instance, taking 60% of recall, the transformation approach achieved 100% of precision, while weighting was around 30% of precision. The results on each cycle, using *SIFT*, TF and $A = 20$, are illustrated on Figure 28(b). Once more, after the fifth cycle low improvements are achieved.



(a) Cycle 10. (b) TF, $A = 20$.

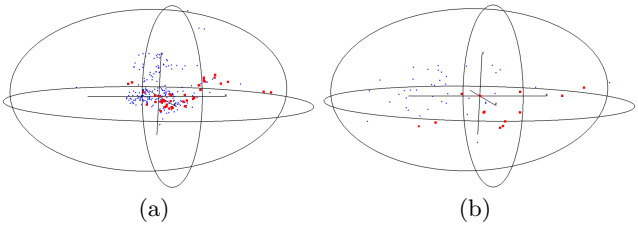
Figure 28: P&R graphics for the feature extractor *SIFT* (a) cycle 10 for each method and for each value of A, (b) evolution through the cycles 01 to 10 using TF and $A = 20$.

Figure 29(a) shows that, also in this data set, the texture-based feature extractors (*Co-occurence* and *Run Length*) achieved the lowest effectiveness on TF method. Considering the *SIFT*, Figure 29(b) illustrates that the TF method significantly outperforms the others, and WF with $A = 20$ has a slightly better result than the WG method in the last three cycles.



(a) TF, $A = 20$. (b) *SIFT*.

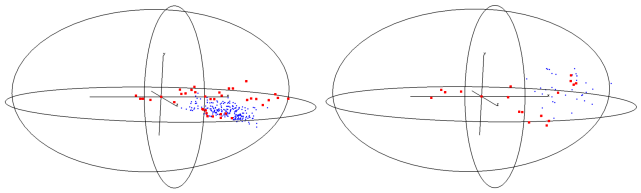
Figure 29: Number of Relevant (a) for each feature extractor using TF and $A = 20$, (b) using *SIFT* for each method and for each value of A.



(a) (b)

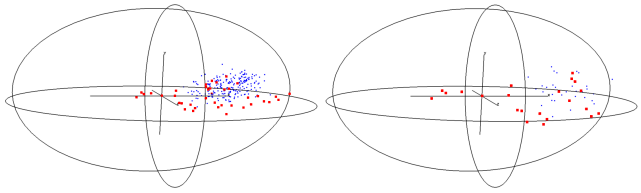
Figure 30: Original space configuration of *Lung* using *SIFT* (a) entire space (b) 50 nearest elements from the query point.

In the visualization and cluster cohesion analysis, *SIFT* was the extractor used in *Lung* data set. Figure 30 shows the distribution of the initial data space. The data spaces obtained from WG, WF and TF methods are illustrated in Figures 31, 32 and 33. In fact, for this data set, all three methods generated a similar distribution, where the relevant elements are more spread than the non relevant elements. Nevertheless, the best configuration after optimization was obtained by TF, since the non relevant images were closer to each other and more distant to the center. SIFT was the best extractor



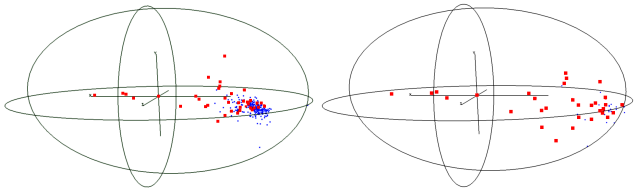
(a) (b)

Figure 31: Data space configuration of *Lung* data set after applying WG method (a) entire space (b) 50 nearest elements from the query point.



(a) (b)

Figure 32: Data space configuration of *Lung* data set after applying WF method (a) entire space (b) 50 nearest elements from the query point.



(a) (b)

Figure 33: Data space configuration of *Lung* data set after applying TF method (a) entire space (b) 50 nearest elements from the query point.

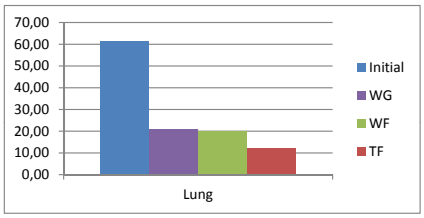


Figure 34: MSD values of clusters of relevant elements on *Lung* data set.

for the lung data set because it can capture the peculiarities of lung images, specially regarding well-defined contours.

Figures 31(b), 32(b), 33(b) show the space with only the 50 closest elements. In the initial space, the number of relevant elements retrieved in the first 50 closest elements retrieved was 13 (Figure 30(b)). The best result after optimization was obtained by TF, with 31 relevant elements, followed by WF with 22 and WG with 18 relevant elements retrieved. Figure 34 shows that, once more, TF had the best cohesion on the relevant elements, and WF was slightly better than WG. It can be observed that all optimization methods generated clusters of relevant elements significantly superior in comparison to the initial space.

ACKNOWLEDGMENTS

The authors are grateful for the financial support granted by CNPq, FAPESP, CAPES, and Microsoft Research.

6. CONCLUSIONS

In this study we have employed Genetic Algorithm techniques combined with Relevance Feedback techniques to improve the accuracy of CBIR systems; we followed this course of action according to two different novel approaches. The first one (WF) infers a weight vector to adjust the Euclidean dissimilarity function by means of weighting functions; the second one (TF) aims at optimizing the features space using linear and non linear transformation functions.

We performed several experiments using images from a general domain and from the medical domain. The results considering feature space transformation functions achieved successful effectiveness, obtaining the highest precision in all experiments, outperforming the other methods by up to 70%. The higher performance obtained by using the features space transformation is due to the fact that it leads to a configuration in which the space is transformed by different functions such as polynomials with several different degrees. Meanwhile, weighted distance functions limit the configuration to linear transformations.

The weighting function approach was more effective than the simple weighting in continuous interval. The advantage of using weighting functions instead of directly generate weights is that the search space is considerably reduced.

We have used visual data analysis in order to visualize the features spaces so to demonstrate that method TF generated space configurations that better express the semantic related to the user's interests; in such visualizations, the clusters of the relevant elements of the queries were grouped closer to the center, as expected. As future work, one possible approach is to investigate the use of the weighting and transformation functions for further feature extractors and distance functions, as well as to study possible correlations between the different feature extractors and their behavior.

7. REFERENCES

[1] Y. Alemu, J. bin Koh, M. Ikram, and D.-K. Kim. Image retrieval in multimedia databases: A survey. In *Proceedings of the Fifth International Conference on Intelligent Information Hiding and Multimedia Signal Processing*, pages 681–689, Los Alamitos, CA, USA, 2009.

[2] S. Brandt. Use of shape features in content-based image retrieval. Master's thesis, Helsinki University of Technology, 1999.

[3] P. Bugatti, A. Traina, and C. Traina. Improving content-based retrieval of medical images through dynamic distance on relevance feedback. In *Proceedings 24th International Symposium on Computer-Based Medical Systems(CBMS)*, pages 1–6, Bristol, IK, 2011.

[4] P. H. Bugatti, A. J. M. Traina, and C. Traina-Jr. Assessing the best integration between

distance-function and image-feature to answer similarity queries. In *Proceedings of the 2008 ACM Symposium on Applied Computing (SAC)*, pages 1225–1230, Fortaleza, Ceara, Brazil, 2008.

[5] S. F. da Silva, M. X. Ribeiro, J. do E.S. Batista Neto, C. Traina-Jr., and A. J. Traina. Improving the ranking quality of medical image retrieval using a genetic feature selection method. *Decision Support Systems*, 51(4):810–820, 2011.

[6] J. dos Santos, C. Ferreira, R. da S. Torres, M. Gonçalves, and R. Lamparelli. A relevance feedback method based on genetic programming for classification of remote sensing images. *Information Sciences*, 181(13):2671 – 2684, 2011.

[7] R. M. Haralick, K. Shanmugam, and I. Dinstein. Textural Features for Image Classification. *IEEE Transactions on Systems, Man and Cybernetics*, 3(6):610–621, 1973.

[8] A. Lakdashti, M. Shahram Moin, and K. Badie. Semantic-based image retrieval: A fuzzy modeling approach. In *Proceedings of the IEEE/ACS International Conference on Computer Systems and Applications*, pages 575–581, Doha, Qatar, 2008.

[9] J. Li and J. Z. Wang. Automatic linguistic indexing of pictures by a statistical modeling approach. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 25:1075–1088, 2003.

[10] H.-H. Loh, J.-G. Leu, and R. Luo. The analysis of natural textures using run length features. *IEEE Transactions on Industrial Electronics*, 35(2):323 –328, 1988.

[11] C. López-Pujalte, V. P. Guerrero-Bote, and F. de Moya-Anegón. Order-based fitness functions for genetic algorithms applied to relevance feedback. *Journal of the American Society for Information Science and Technology*, 54(2):152–160, January 2003.

[12] D. G. Lowe. Object recognition from local scale-invariant features. In *Proceedings of the 7th IEEE International Conference on Computer Vision*, volume 2, pages 1150–1157, Los Alamitos, CA, USA, 1999.

[13] d. l. T. F. Nguyen, M.H. Optimal feature selection for support vector machines. *Pattern Recognition*, 43:584–591, 2010.

[14] A. Oliva and A. Torralba. Modeling the shape of the scene: A holistic representation of the spatial envelope. *International Journal of Computer Vision*, 42:145–175, 2001.

[15] M. Ribeiro, A. Balan, J. Felipe, A. Traina, and C. Traina. Mining statistical association rules to select the most relevant medical image features. In D. Zighed, S. Tsumoto, Z. Ras, and H. Hacid, editors, *Mining Complex Data*, volume 165 of *Studies in Computational Intelligence*, pages 113–131. Springer Berlin / Heidelberg, 2009.

[16] J. F. Rodrigues Jr., L. A. M. Zaina, L. A. S. Romani, and R. R. Ciferri. Metricsplat - a platform for quick development, testing and visualization of content-based retrieval techniques. In *Proceedings of the 24th Brazilian Symposium on Databases*, Fortaleza, Ceara, Brazil.

[17] A. W. M. Smeulders, M. Worring, S. Santini, A. Gupta, and R. Jain. Content-based image retrieval at the end of the early years. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(12):1349–1380, 2000.

[18] Z. Stejic, Y. Takama, and K. Hirota. Genetic algorithms for a family of image similarity models incorporated in the relevance feedback mechanism. *Applied Soft Computing*, 2(4):306 – 327, 2003.

[19] M. A. Stricker and M. Orengo. Similarity of color images. In *Proceedings of the Storage and Retrieval for Image and Video Databases (SPIE*, pages 381–392, San Diego CA, USA, 1995.

[20] J.-H. Su, W.-J. Huang, P. S. Yu, and V. S. Tseng. Efficient relevance feedback for content-based image retrieval by mining user navigation patterns. *IEEE Transactions on Knowledge and Data Engineering*, 23(3):360–372, 2011.

[21] R. d. S. Torres, A. X. Falcão, M. A. Gonçalves, J. a. P. Papa, B. Zhang, W. Fan, and E. A. Fox. A genetic programming framework for content-based image retrieval. *Pattern Recognition*, 42:283–292, February 2009.

[22] P. Zezula, G. Amato, V. Dohnal, and M. Batko. *Similarity Search: The Metric Space Approach*, volume 32 of *Advances in Database Systems*. Springer, 2006.

[23] X. S. Zhou and T. S. Huang. Relevance feedback in image retrieval: A comprehensive review. *Multimedia Systems*, 8(6):536–544, 2003.

ABOUT THE AUTHORS:

Letricia P. S. Avalhais received the MSc degree in Computer Science and Computational Mathematics at University of Sao Paulo (in January 2012), and the BSc in Computer Science at Federal University of Mato Grosso do Sul (in December 2008). She is currently a Ph.D. candidate at University of Sao Paulo. Her research interests include Image Processing, Content-Based Image Retrieval, Video Analysis, Visualization and Machine Learning.

Sergio F. Silva received the B.Sc. degree in computer science from the Federal University of Goias, Brazil, in 2004. He received the the M.Sc. degree in computer science at the Faculty of Computation of the University of Uberlandia, Brazil, in 2007 and Ph.D. degree in computer science at the Mathematics and Computer Science Institute, University of Sao Paulo at Sao Carlos, Brazil. His research interests include multimedia data mining, computer-aided diagnosis and content-based image retrieval with special attention for optimization techniques based on evolutionary computation.

Jose F. Rodrigues Jr. is a Professor at University of Sao Paulo, Brazil. He received his Ph.D. from this same university, part of which was carried out at Carnegie Mellon University in 2007. Jose Fernando is a regular author and reviewer of major events in his field, having contributed with publications in IEEE and ACM journals and conferences. His topics of research include data analysis, content-based data retrieval, and visualization.

Agma J. M. Traina received the B.Sc., the M.Sc. and Ph.D. degrees in computer science from the University of Sao Paulo, Brazil, in 1983, 1987 and 1991, respectively. She is currently a full Professor with the Computer Science Department of the University of Sao Paulo at Sao Carlos, Brazil. Her research interests include image databases, image mining, indexing methods for multidimensional data, information visualization, image processing for medical applications and optimization techniques based on evolutionary computation.

Caetano Traina Jr. received the B.Sc. degree in electrical engineering, the M.Sc. and Ph.D. degrees in computer science from the University of Sao Paulo, Brazil, in 1978, 1982 and 1987, respectively. He is currently a full professor with the Computer Science Department of the University of Sao Paulo at Sao Carlos, Brazil. His research interests include access methods for complex data, data mining, similarity searching and multimedia databases.

An Empirical Study on Clone Stability

Manishankar Mondal Chanchal K. Roy Kevin A. Schneider
Department of Computer Science,
University of Saskatchewan, Canada
{mshankar.mondal, chanchal.roy, kevin.schneider}@usask.ca

ABSTRACT

Code cloning is a controversial software engineering practice due to contradictory claims regarding its effect on software maintenance. Code stability is a recently introduced measurement technique that has been used to determine the impact of code cloning by quantifying the changeability of a code region. Although most existing stability analysis studies agree that cloned code is more stable than non-cloned code, the studies have two major flaws: (i) each study only considered a single stability measurement (e.g., lines of code changed, frequency of change, age of change); and, (ii) only a small number of subject systems were analyzed and these were of limited variety.

In this paper, we present a comprehensive empirical study on code stability using four different stability measuring methods. We use a recently introduced hybrid clone detection tool, NiCAD, to detect the clones and analyze their stability in different dimensions: by clone type, by measuring method, by programming language, and by system size and age. Our in-depth investigation on 12 diverse subject systems written in three programming languages considering three types of clones reveals that: (i) cloned code is generally less stable than non-cloned code, and more specifically both Type-1 and Type-2 clones show higher instability than Type-3 clones; (ii) clones in both Java and C systems exhibit higher instability compared to the clones in C# systems; (iii) a system's development strategy might play a key role in defining its comparative code stability scenario; and, (iv) cloned and non-cloned regions of a subject system do not follow any consistent change pattern.[1]

Categories and Subject Descriptors

D.2.7 [**Software Engineering**]: Distribution, Maintenance, and Enhancement—*Restructuring, Reverse Engineering and Reengineering.*

General Terms

Measurement and Experimentation

Keywords

Software Clones, Types of Clones, Code Stability, Modification Frequency, Changeability, Overall Instability.

[1]This work is based on an earlier work: SAC '12 Proceedings of the 2012 ACM Symposium on Applied Computing, Copyright 2012 ACM 978-1-4503-0857-1/12/03. http://doi.acm.org/10.1145/2245276.2231969.

1. INTRODUCTION

Frequent copy-paste activity by programmers during software development is common. Copying a code fragment from one location and pasting it to another location with or without modifications cause multiple copies of exact or closely similar code fragments to co-exist in software systems. These code fragments are known as clones [22, 25]. Whatever may be the reasons behind cloning, the impact of clones on software maintenance and evolution is of great concern.

The common belief is that the presence of duplicate code poses additional challenges to software maintenance by making inconsistent changes more difficult, introducing bugs and as a result increasing maintenance efforts. From this point of view, some researchers have identified clones as "bad smells" and their studies showed that clones have negative impact on software quality and maintenance [8, 15, 16, 19]. On the other hand, there has been a good number of empirical evidence in favour of clones concluding that clones are not harmful [1, 7, 10, 11, 27]. Instead, clones can be useful from different points of views [9].

A widely used term to assess the impact of clones on software maintenance is *stability* [7, 12, 13, 15]. In general, *stability of a particular code region measures the extent to which that code region remains stable (or unchanged) during the evolution of a software system.* If cloned code is more stable (changes less frequently) as compared to non-cloned code during software evolution, it can be concluded that cloned code does not significantly increase maintenance efforts. Different studies have defined and evaluated stability from different viewpoints which can be broadly divided into two categories:

(1) Stability measurement in terms of changes: Some methodologies [7, 12, 15, 6] have measured stability by quantifying the changes to a code region using two general approaches: (i) determination of the ratio of the number of lines added, modified and deleted to the total number of lines in a code region (cloned or non-cloned) [12, 15, 6] and (ii) determination of the frequency of modifications to the cloned and non-cloned code [7] with the hypothesis that the higher the modification frequency of a code region is the less stable it is.

(2) Stability measurement in terms of age: This approach [13] determines the average last changed dates of cloned and non-cloned code of a subject system. The hypothesis is that the older the average last change date of a code region is, the more stable it is.

To measure the comparative stability of cloned and non-cloned code, Krinke carried out two case studies [12, 13].

In his first study, he calculated the comparative instabilities caused by cloned and non-cloned regions in terms of addition, deletion and modification in these regions whereas in a most recent study [13] (elaborated in Section 3.2), he determined the average last change dates of the regions. Both of these studies suggest cloned code to be more stable than non-cloned code.

Hotta et al., in a recent study [7], calculated the modification frequencies of cloned and non-cloned code and found that the modification frequency of non-cloned code is higher than that of cloned code.

1.1 Motivation

Though all of the studies [7, 12, 13, 6] generally agreed on the higher stability of cloned code over non-cloned code, we conducted our research to address the following drawbacks of these studies.

(1) The studies only considered limited aspects of stability.

(2) General decisions were made without considering a wide variety of subject systems.

(3) No study addresses the comparative stabilities as well as impacts of different clone types. This issue is important in the sense that different types of clones might have different impacts (good or bad) on maintenance. Based on impact variability, we might take care of some specific clone types while leaving others alone.

(4) Instability of clones on the basis of language variability was not measured. This information might be very important from a managerial perspective. Projects developed using programming languages that exhibit high clone instability may require more care regarding cloning activities.

(5) Different studies were conducted on different experimental setups (e.g. different subject systems, different clone detection tools with different parameters, considering software releases or revisions at different intervals), which might be a potential cause behind their contradictory outcomes.

(6) No study performed statistical tests about how significant is the difference between the metric values of cloned and non-cloned code. If the metric values regarding cloned and non-cloned code are not significantly different, then we do not need to be much worried about clones.

(7) The existing metrics are not sufficient to reveal all aspects of changeability (as well as stability) of cloned and non-cloned code.

1.2 Contribution

Focusing on the issues mentioned above, our study contributes in the following ways.

(1) Considering the count of lines affected in source code modifications (additions, deletions or changes) we propose a new method which calculates four new metrics: UP (Unstable Proportion), UPHL (Unstable Proportion per 100 LOC), PCRM (Proportion of Code Region Modified), and OICR (Overall Instability of Code Region) that provide us more precise information about code changeability in comparison with existing metrics as well as methods [12, 6] that considered line counts. The comparison between our proposed metrics and the existing ones has been elaborated in Section 3.5. We investigated these metrics to compare the stability of cloned and non-cloned code.

(2) We have investigated four methods in total by implementing them on the same experimental setup and answered seven research questions as listed in Table 1. One of the four methods is our proposed new method that has already been mentioned in the previous point. Two of these methods are existing and were proposed by Hotta et al. [7] and Krinke [13]. The last method is our proposed variant of Krinke's method [13]. The reasons behind introducing this variant are elaborated in Section 3.3. The research questions (Table 1) belong to five different dimensions. The first three research questions are answered by investigating the metrics calculated by our proposed new method. The other four questions are answered by combining the experimental results of all four candidate methods.

For detecting clones, we used the recently introduced hybrid clone detection tool NiCad [3] that combines the strengths and overcomes the limitations of both text-based and AST-based clone detection techniques and exploits novel applications of a source transformation system to yield highly accurate identification of Type-1, Type-2 and Type-3 cloned code in software systems [20].

Our experimental results on three clone types of 12 subject systems written in three languages (Java, C and C#) reveal that:

(1) Cloned code gets modified significantly more often (supported with Mann-Whitney Wilcoxon (MWW) tests [17]) than non-cloned code.

(2) A significantly higher proportion of cloned LOC is modified in commit operations compared to non-cloned LOC.

(3) Type-1 and Type-2 clones are potential threats to a system's stability while Type-3 clones are possibly not.

(4) Clones in Java and C systems exhibit a higher level of instabilities as compared to those of C# systems. This is also statistically supported by Fisher's Exact Test [5].

(5) Cloned code generally exhibits higher instability than non-cloned code. However, cloned and non-cloned regions of subject systems do not follow any consistent change pattern. Moreover, the development strategy may have a strong impact on the stability of cloned and non-cloned code.

The rest of the paper is organized as follows. Section 2 outlines the relevant research. Section 3 elaborates on the candidate methods. Our experimental setup is described in Section 4 and Section 5 contains the experimental results. A detailed analysis of the experimental results is presented in Section 6 while Section 7 mentions some possible validity threats of our study. Section 8 concludes the paper and describes future directions. The work presented in this paper is an extended version of our earlier work [18].

2. RELATED WORK

Over the last several years, the impact of clones has been an area of focus for software engineering research resulting in a significant number of studies and empirical evidence. Kim et al. [10] introduced a clone genealogy model to study clone evolution and applied the model on two medium sized Java systems. They showed that: (i) refactoring of clones

Table 1: Research Questions

	Research Questions (RQs)	Dimensions
RQ1	How often is a particular code region type (cloned or non-cloned) modified during evolution? Which code region type is modified more often?	Comparative stability centric
RQ2	Which code region type (cloned or non-cloned) has a higher proportion of LOC modifications in the commit operations?	
RQ3	Which code region type (cloned or non-cloned) exhibits higher instability compared to the other?	
RQ4	Do different types of clones exhibit different stability scenarios? Which type(s) of clones is (are) more vulnerable to the system's stability?	Type centric
RQ5	Why and to what extent do the decisions made by different methods on the same subject system differ?	Method centric
RQ6	Do different programming languages exhibit different stability scenarios?	Language centric
RQ7	Is there any effect of system sizes and system ages on the stability of cloned and non-cloned code?	System centric

may not always improve software quality and (ii) immediate refactoring of short-lived clones is not required and that such clones might not be harmful. Saha et al. [27] extended their work by extracting and evaluating code clone genealogies at the release level of 17 open source systems and reported that most of the clones do not require any refactoring effort in the maintenance phase.

Kapser and Godfrey [9] strongly argued against the conventional belief of harmfulness of clones by investigating different cloning patterns. They showed that: (i) about 71% of the cloned code has a kind of positive impact in software maintenance and (ii) cloning can be an effective way of reusing stable and mature features in software evolution. Lozano and Wermelinger et al. performed three studies [14, 15, 16] on the impact of clones on software maintenance considering method level granularities using CCFinder [2]. They developed a prototype tool *CloneTracker* [14] to investigate the changeability of clones. The other studies, though conducted on a small number of Java systems (4 in [15] and 5 in [16]), reported that clones have a harmful impact on the maintenance phase because clones often increase maintenance efforts and are vulnerable to a system's stability.

Juergens et al. [8] studied the impact of clones on large scale commercial systems and suggested that: (i) inconsistent changes occurs frequently with cloned code and (ii) nearly every second unintentional inconsistent change to a clone leads to a fault. Aversano et al. [1] on the other hand, carried out an empirical study that combines the clone detection and co-change analysis to investigate how clones are maintained during evolution or bug fixing. Their case study on two subject systems confirmed that most of the clones are consistently maintained. Thummalapenta et al. [28] in another empirical study on four subject systems reported that most of the clones are changed consistently and other inconsistently changed fragments evolve independently.

In a recent study [6] Göde et al. replicated and extended Krinke's study [12] using an incremental clone detection technique to validate the outcome of Krinke's study. He supported Krinke by assessing cloned code to be more stable than non-cloned code in general.

Code stability related research conducted by Krinke [12, 13] and Hotta et al. [7] referred to in the introduction is elaborated on in the next section.

In our empirical study, we have replicated Krinke's [13] and Hotta et al.'s [7] methods and implemented our variant of Krinke's method [13] and our proposed new method using NiCad [3]. Our experimental results and analysis of those results reveal inportant information about comparative stabilities and harmfulness of three clone types along with language based stability trends.

3. STABILITY MEASURING METHODS

This section discusses the three methods and associated metrics that we have implemented for our investigation. These methods follow different approaches and calculate different metrics but their aim is identical in the sense that each of these methods takes the decision about whether cloned code of a subject system is more stable than its non-cloned code. For this reason we perform a head-to-head comparison of the stability decisions indicated by the metrics of these three methods and focus on the implementation and strategic differences that cause decision dissimilarities.

3.1 Modification Frequencies

Hotta et al. [7] calculated two metrics: (i) MF_d (Modification Frequencies of Duplicate code) and (ii) MF_n (Modification Frequencies of Non-Duplicate code) considering all the revisions of a given codebase extracted from SVN. Their metric calculation strategy involves identification and checking out of relevant revisions of a subject system, normalization of source files by removing blank lines, comments and indents, detection and storing of each line of duplicate code into the database. The differences between consecutive revisions are also identified and stored in the database. Then, MC_d (Modification Count in Duplicate code region) and MC_n (Modification Count in Non-Duplicate code region) are determined exploiting the information saved in the database and finally MF_d and MF_n are calculated using the following equations [7]:

$$MF_d = \frac{\sum_{r \epsilon R} MC_d(r)}{|R|} * \frac{\sum_{r \epsilon R} LOC(r)}{\sum_{r \epsilon R} LOC_d(r)} \quad (1)$$

$$MF_n = \frac{\sum_{r \epsilon R} MC_n(r)}{|R|} * \frac{\sum_{r \epsilon R} LOC(r)}{\sum_{r \epsilon R} LOC_n(r)} \quad (2)$$

Here, R is the number of revisions of the candidate subject system. $MC_d(r)$ and $MC_n(r)$ are the number of modifications in the duplicate and non-duplicate code regions respec-

tively between revisions r and $(r+1)$. MF_d and MF_n are the modification frequencies of the duplicate and non-duplicate code regions of the system. $LOC(r)$ is the number of LOC in revision r. $LOC_d(r)$ and $LOC_n(r)$ are respectively the numbers of duplicate and non-duplicate LOCs in revision r.

3.2 Average Last Change Date

Krinke [13] introduced a new concept of code stability measurement by calculating the average last change dates of cloned and non-cloned regions of a codebase using the *blame* command of SVN. He considers only a single revision (generally the last revision) unlike the previous method proposed by Hotta et al. [7] that considers all the revisions up to the last one. The *blame* command on a file retrieves each line's revision and date when the line was last changed. He calculates the average last change dates of cloned and non-cloned code from the file level and system level granularities.

File level metrics: (1) Percentage of files where the average last change date of cloned code is older than that of non-cloned code (PF_c) (cloned code is older than non-cloned code) in the last revision of a subject system. **(2)** Percentage of files where the average last change date of cloned code is newer than that of non-cloned code (PF_n) (cloned code is younger than non-cloned code) in the last revision of a subject system.

System level metrics: (1) Average last change date of cloned code (ALC_c) for the last revision of a candidate subject system. **(2)** Average last change date of non-cloned code (ALC_n) for the last revision of a candidate subject system. To calculate file level metrics in our implementation, we considered only the analyzable source files, which excludes two categories of files from consideration: (i) files containing no cloned code and (ii) fully cloned files. But, system level metrics are calculated considering all source files. According to this method, the older the code is the more stable it is.

Calculation of average last change date: Suppose five lines in a file correspond to 5 revision dates (or last change dates) 01-Jan-11, 05-Jan-11, 08-Jan-11, 12-Jan-11, 20-Jan-11. The average of these dates was calculated by determining the average distance (in days) of all other dates from the oldest date 01-Jan-11. This average distance is $(4+7+11+19)/4 = 10.25$ and thus the average date is 10.25 days later to 01-Jan-11 yielding 11-Jan-11.

3.3 Proposed Variant of Krinke's Method

We have proposed a variant of Krinke's methodology [13] to analyze the longevity (stability) of cloned and non-cloned code by calculating their average ages. We also have used the *blame* command of SVN to calculate the age for each of the cloned and non-cloned lines in a subject system.

Suppose we have several subject systems. For a specific subject system we work on its last revision R. By applying a clone detector on revision R, we can separate the lines of each source file into two disjoint sets: (i) containing all cloned lines and (ii) containing all non-cloned lines. Different lines of a file contained in R can belong to different previous revisions. If the *blame* command on a file assigns the revision r to a line x, then we understand that line x was produced in revision r and has not been changed up to the last revision R. We denote the revision of x as $r = revision(x)$. The creation date of r is denoted as $date(r)$. In the last revision R, we can determine the age (in days) of this line by the following equation:

$$age(x) = date(R) - date(revision(x)) \quad (3)$$

We have calculated the following two average ages for cloned and non-cloned code from system level granularity.

1. Average age of cloned code (AA_c) in the last revision of a subject system. This is calculated by considering all cloned lines of all source files of the system.

2. Average age of non-cloned code (AA_n) in the last revision of a subject system. AA_n is calculated by considering all non-cloned lines of all source files of the system.
According to our method, a higher average age is the implication of higher stability.

We have introduced this variant to address the following issues in Krinke's method.

1. *blame* command of SVN gives the revisions as well as revision dates of all lines of a source file including its comments and blank lines. Krinke's method does not exclude blank lines and comments from consideration. This might play a significant role on skewing the real stability scenario.

2. As indicated in the average last change date calculation process, Krinke's method often introduces some rounding errors in its results. This might force the average last change dates of cloned and non-cloned code to be equal (There are examples in Section 5).

3. The method's dependability on the file level metrics sometimes alters the real stability scenario. The type-3 case of 'Greenshot' is an example where both Hotta et al.'s method and our proposed variant make a similar decision (non-cloned code is more stable); but, the file level metrics of Krinke's method alters this decision. The system level metrics (ALCs) of Krinke's method could not make a stability determination because the metrics corresponding to cloned (ALC_c) and non- cloned (ALC_n) code were the same.

Our proposed variant overcomes these issues while calculating stability results. It does not calculate any file level metrics because its system level metrics are adequate in decision making. It should also be mentioned that Hotta et al.'s method also ensures the exclusion of blank lines and comments from consideration through some preprocessing steps prior to clone detection.

3.4 Proposed New Method and Metrics

Hotta et al.'s method [7] calculates modification frequency by only considering the count of modifications that occurred in a subject system without considering the number of lines affected by those modifications. Focusing on this issue, we propose a new method that calculates four new metrics for measuring the stability (as well as changeability) of a particular code region. The descriptions and calculation procedures of the metrics are given below.

UP (Unstable Proportion): Unstable proportion (UP) of a particular code region (cloned or non-cloned) is the proportion of the commit operations in which that code region is modified.

Suppose C is the set of all commit operations through which a subject system has evolved. The two sets of commit op-

erations in which the cloned and non-cloned regions of this system was modified are C_c and C_n respectively. We should note that the sets C_c and C_n might not be disjoint. The unstable proportions of cloned and non-cloned regions are calculated using the following two equations, respectively.

$$UP_c = \frac{100 \times |C_c|}{|C|} \quad (4)$$

$$UP_n = \frac{100 \times |C_n|}{|C|} \quad (5)$$

Here, UP_c is the unstable proportion of cloned code and UP_n is the unstable proportion of non-cloned code.

UPHL (Unstable Proportion per 100 LOC): Generally the UP of a particular code region (cloned or non-cloned) is positively proportional to the total LOC of that region. UP_n is expected to be higher than the corresponding UP_c for a particular subject system. To determine how often a particular code region is modified, we should also consider the total LOC of that region. From this perspective we calculated the *UPHL* using equations Eq. 6 and Eq. 7. According to the equations, the *UPHL* of a particular region is equal to the UP per 100 LOC of that region. In other words, *UPHL determines the likelihood of being modified per 100 LOC of a code region.* As there are many revisions of a particular software system, we determined the total LOC of the cloned or non-cloned region of this system by calculating the average LOC per revision of the corresponding region.

$$UPHL_c = \frac{100 \times UP_c \times |C|}{\sum_{c_i \epsilon C} LOC_c c_i} \quad (6)$$

$$UPHL_n = \frac{100 \times UP_n \times |C|}{\sum_{c_i \epsilon C} LOC_n c_i} \quad (7)$$

Here, $UPHL_c$ and $UPHL_n$ are the unstable proportions per 100 LOC of the cloned and non-cloned regions respectively. $LOC_c(c_i)$ is the count of total cloned LOC of the revision corresponding to the commit c_i. Also, $LOC_n(c_i)$ is the count of total non-cloned LOC of the revision corresponding to the commit c_i.

PCRM (Proportion of Code Region Modified): For a particular code region (cloned or non-cloned) we calculate the PCRM by determining the proportion of that code region getting modified in commit operations. Here, we consider only those commit operations where the particular code region had some modifications. Considering the previous example, we can calculate the *PCRM* of cloned and non-cloned code according to the following equations.

$$PCRM_c = \frac{100 \times \sum_{c_i \epsilon C_c} LC_c(c_i)}{\sum_{c_i \epsilon C_c} LOC_c(c_i)} \quad (8)$$

$$PCRM_n = \frac{100 \times \sum_{c_i \epsilon C_n} LC_n(c_i)}{\sum_{c_i \epsilon C_n} LOC_n(c_i)} \quad (9)$$

Here, $PCRM_c$ and $PCRM_n$ are respectively the proportions of cloned and non-cloned regions that are modified. $LC_c(c_i)$ and $LC_n(c_i)$ are the number of lines changed in cloned and non-cloned regions in commit operation c_i.

OICR (Overall instability of code region): We calculate the OICR of a particular code region (cloned or non-cloned) by multiplying its UP with its PCRM. We see that the PCRM determines the proportion of a particular code region being modified per commit operation and UP determines the proportion of commit operations in which that particular code region is being modified. Thus, the multiplication of these two will determine the instability exhibited by the code region throughout the evolution. We calculate OICR for cloned and non-cloned code according to the following two equations.

$$OICR_c = UP_c \times PCRM_c \quad (10)$$

$$OICR_n = UP_n \times PCRM_n \quad (11)$$

Here, $OICR_c$ and $OICR_n$ are respectively the overall instabilities of cloned and non-cloned regions of a software system.

3.5 Comparison of Our Proposed Metrics With Existing Ones

Two existing studies performed by Krinke [12] and Göde and Harder [6] investigated some metrics that are similar to our proposed metric PCRM. Krinke [12] computed the instabilities of cloned and non-cloned code with respect to addition, deletion, and change. Göde and Harder extended Krinke's study [12] considering tokens instead of lines. While Göde and Harder analyzed all of the revisions of a particular software system, Krinke considered 200 weekly snapshots for each of his candidate systems. However, none of these studies could answer the first and second research questions (RQ 1 and RQ 2 in Table 1).

RQ 1 was not addressed because no existing study defined and evaluated a relevant metric. Our proposed metric *UPHL* addresses this question.

Also, RQ 2 was not addressed by any existing studies. Our metric *PCRM* answers this question. The studies performed by Krinke [12] and Göde and Harder [6] computed some related metrics. However, the strategic difference between our proposed metric (PCRM) and the existing ones is that while calculating the metric value for a particular code region we considered only those commits where that particular code region was modified excluding those commits where that region did not have any modifications. However, the existing studies did not exclude commits that do not have any effect on a particular code region while calculating the metric value for the region.

We think that if a particular commit does not affect a particular code region, we should not consider that commit for investigating the stability of that region, because that commit is not responsible for increasing the instability of that region. We call such a commit an *ineffective commit* for that particular region. As the previous studies [12, 6] did not exclude these ineffective commits for a particular region while investigating the region's instability, no study could determine what proportion of code from a particular region is being modified when that region is actually receiving some changes (due to effective commits). Our proposed metric PCRM eliminates this drawback of the similar existing metrics.

3.6 Major Difference Between Hotta's Method and Our Proposed New Method

Hotta et al.'s method relies on the count of modifications for making stability decisions disregarding the number of lines affected by the modifications. However, our proposed new method relies on the count of affected lines. In a particular commit operation, l ($l > 0$) consecutive lines of a particular code region might get modified. According to Hotta et al.'s method the count of modifications is one. It calculates modification frequencies by considering this modification count disregarding the number of lines modified. However, our proposed new method calculates PCRM and OICR considering the count of affected lines (for this example l).

This difference may cause disagreements in the stability decisions made by the two methods. Suppose the cloned and non-cloned regions of a software system received respectively m_c and m_{nc} modifications in a commit operation. The total number of lines affected by these m_c and m_{nc} modifications are l_c and l_{nc} respectively. If $m_c > m_{nc}$, Hotta et al.'s method will decide that cloned code is more unstable. However, in this situation our proposed new method may decide the opposite, since l_{nc} could be greater than l_c.

3.7 Major Difference Between Hotta's Method and the Method Proposed by Krinke and Its Variant

Hotta et al.'s method [7] considers *all* modifications to a region from its creation and it does not matter *when* the modifications to the region are applied. The other two methods only consider the *last* modification (which can also be creation) and do not consider any modification before.

Suppose a file contains two lines denoted by x and y in revision 1 and this file passed through 100 commits during which x had 5 changes and y had only one change. Let the change on y occur at the 99th commit and the last change on x occur at the 50th commit. A *blame* command on the last revision (100) of this file will assign x revision 50 and y will be assigned revision 99. According to both Krinke's method and our variant, x is older than y because the revision date corresponding to revision 50 is much older than the revision date corresponding to revision 99 and thus, x will be suggested to be more stable than y by these two methods. On the other hand, the method proposed by Hotta et al. counts the number of modifications that occurred on these two lines. Consequently, Hotta et al. will suggest y to be more stable than x because the modification frequency of x will obviously be greater than that of y.

4. EXPERIMENTAL SETUP

We implemented all four candidate methods in a common framework in Java using MySQL for the back-end database. Instead of using any existing implementations, we have reimplemented the two already existing methods (proposed by Hotta et al.[7] and Krinke [13]) as we wanted to have a common framework for comparison. Our selected subject systems and setup of the clone detection tool are described below.

4.1 Clone Detection

We used the NiCad [3, 21] clone detection tool to detect clones in the subject systems in our study. NiCad can detect both exact and near-miss clones at the function or block level of granularity. We detected block clones with a minimum size of 5 LOC in the pretty-printed format that removes comments and formatting differences.

NiCad can provide clone detection results in two ways: (1) by separating three types of clones (Type-1, Type-2, Type-3) and (2) by combining all three types of clones. The NiCad settings for detecting the three types of clones is provided in Table 2. The dissimilarity threshold means that the clone fragments in a particular clone class may have dissimilarities up to that particular threshold value between the pretty-printed and/or normalized code fragments. We set the dissimilarity threshold to 20% with blind renaming of identifiers for detecting Type-3 clones. For all these settings above NiCad was shown to have high precision and recall [20]. We have used NiCad's combined type results for answering the first three research questions. The remaining four questions have been answered by investigating the three types of clones separately.

Table 2: NiCad Settings

Clone Types	Identifier Renaming	Dissimilarity Threshold
Type 1	none	0%
Type 2	blindrename	0%
Type 3	blindrename	20%

Table 3: Subject Systems

	Systems	Domains	LOC	Rev
Java	DNSJava	DNS protocol	23,373	1635
	Ant-Contrib	Web Server	12,621	176
	Carol	Game	25,092	1699
	Plandora	Project Management	79,853	32
C	Ctags	Code Def. Generator	33,270	774
	Camellia	Image Processing	100,891	608
	QMail Admin	Mail Management	4,054	317
	Hashkill	Password Cracker	83,269	110
C#	GreenShot	Multimedia	37,628	999
	ImgSeqScan	Multimedia	12,393	73
	Capital Resource	Database Management	75,434	122
	MonoOSC	Formats and Protocols	18,991	355

Rev = Revisions

4.2 Subject Systems

Table 3 lists the details of the subject systems used in our study. We selected these subject systems because they are diverse in nature, differ in size, span 11 application domains, and are written in three programming languages. Also, most of these systems differ from those included in the studies of Krinke[13] and Hotta et al.[7], which was intentionally done to retrieve exact stability scenarios.

5. EXPERIMENTAL RESULTS

For answering the first three research questions we applied our proposed new method on the combined type clone detec-

Table 4: Decision Making Strategy

Method	Metrics		Decision Making	
	CC	NC	CC More Stable	NC More Stable
Proposed Method	$OICR_c$	$OICR_n$	$OICR_c$ $<OICR_n$	$OICR_n$ $<OICR_c$
Hotta et al. [7]	MF_d	MF_n	MF_d $<MF_n$	MF_n $<MF_d$
Krinke [13]	ALC_c, PF_c	ALC_n, PF_n	ALC_c is older	ALC_n is older
Krinke's Variant	AA_c	AA_n	AA_c $>AA_n$	AA_n $>AA_c$
Special Decision for Krinke's Method				
if $ALC_c = ALC_n$ then				
$PF_c > PF_n$ implies CC more stable				
$PF_n > PF_c$ implies NC more stable				
CC = Cloned Code		NC = Non-cloned Code		

tion results of each of the 12 subject systems and obtained the values of the four new metrics. However, for answering the remaining questions we applied each of the four methods on each of the three types of clones of each of the 12 candidate systems and calculated the metric values for three types of clones separately. By applying our proposed new method on the individual type results we obtained the value of the fourth metric (OICR) only. So, in our investigation regarding the research questions 4 to 7, each of the four methods contributed one metric (4 metrics in total).

We should mention that we have three different implementations corresponding to three types of clones for each of the candidate methods. Thus, we have 12 (4 subject systems × 3 clone types) stability evaluation systems in total. For answering the RQs 4 to 7, we applied each of these 12 stability evaluation systems on each of the 12 subject systems to get the values of the stability metrics. So, we have 144 (12 subject systems × 12 stability evaluation systems) sets of metric values from 144 different experiments. Each set contains two values: (1) the metric value for cloned code (of a particular type), and (ii) the metric value for non-cloned code (corresponding to that particular type). From these two values, we can make a decision about comparative stability of cloned and non-cloned code. For this reason, we have called each of these metric value sets a decision point. Finally, our investigation regarding the RQs 4 to 7 depends on these 144 decision points obtained by conducting 144 different experiments. However, for answering the RQs 1 to 3 we conducted 12 experiments (by applying our proposed new method on the combined type results of 12 subject systems). The following paragraphs in this section describes the tables that contain the results obtained from the experiments.

Table 5 shows the average last change dates obtained by applying Krinke's method. Table 7 and Table 9 contain respectively the modification frequencies and average ages of cloned and non-cloned code. File level metrics for two special cases (Table 4) are shown in Table 6. The overall instabilities of cloned and non-cloned code obtained by applying our proposed new method are presented in Table 8. Interpretation of the table data is explained below.

Almost all of the tables are self-explanatory. Decision making strategies for Tables 5, 7, 9, and 8 are elaborated in Table 4. However, for our proposed new method we have specified only one metric *Overall Instability of Code Region* (out of four) in Table 4. The other three metrics are investigated in section 6.1.

Table 6: File Level Metrics for Two Systems

Subject System	Clone Type	PF_c	PF_n
Plandora	Type-2	6	4
Greenshot	Type-3	43	12

Table 7: Modification Frequencies of Cloned (MF_d) and Non-cloned (MF_n) code by Hotta et al.'s method

	Systems	Type 1		Type 2		Type 3	
		MF_d	MF_n	MF_d	MF_n	MF_d	MF_n
Java	DNSJava	21.61	7.12	19.34	6.99	7.93	8.66
	Ant-Contrib	3.62	1.49	2.02	1.52	1.43	1.59
	Carol	8.15	6.60	4.07	3.69	9.91	8.97
	Plandora	0.44	0.92	0.45	0.97	0.55	1.11
C	Ctags	6.37	3.82	7.19	7.17	6.71	3.68
	Camellia	18.50	18.04	42.37	17.73	30.02	17.53
	QMailAdmin	5.09	2.74	8.83	5.47	8.24	2.58
	Hash Kill	61.24	115.22	59.92	115.64	65.75	118.04
C#	GreenShot	7.94	6.07	6.92	6.07	8.13	6.06
	ImgSeqScan	0	20.93	0	21.06	0	21.29
	Capital Resource	0	67.15	0	67.31	3.63	67.11
	MonoOSC	8.58	29.14	7.92	29.23	10.62	29.63

The stability decisions (as per Table 4) of all the metric values contained in the Tables 5, 7, 8, and 9 are summarized in Table 10, which contains decisions for 144 (12 subject systems x 4 methods x 3 clone types) decision points corresponding to 144 cells containing stability decision symbols ('⊕' and '⊖', explained in the table).

For decision making regarding Krinke's method we prioritized the system level metrics (ALC_c and ALC_n) as they represent the exact scenarios of the whole system. There are only two examples of special cases as per Table 4: (i) Type-3 case of 'Greenshot' and (ii) Type-2 case of 'Plandora'. For these, the system level metrics (Table 5) are the same and thus, we based the decisions on the file level metrics. We show the file level metrics for these two cases in Table 6 without providing them for all 36 cases (12 subject systems x 3 clone types).

6. ANALYSIS OF RESULTS

We present our analysis of the experimental results in five dimensions and answer the seven research questions introduced in Table 1.

To address the first three research questions we produced four graphs: Fig. 1, Fig. 2, Fig. 3, and Fig. 4. Table 11 contains 36 (12 subject systems, 3 clone types) decision points and was developed from Table 10 for the purpose of answering research questions 4 to 7. Each cell in the table corresponds to a decision point and implies agreement ('⊕' or '⊖') or disagreement ('⊗') of the candidate methods regarding stability decisions. The meanings of '⊕', '⊖' and '⊗' are provided in the tables.

Table 5: Average Last Change Dates of Cloned (ALC_c) and Non-cloned (ALC_n) code

	Clone Types	Type 1		Type 2		Type 3	
	Systems	ALC_c	ALC_n	ALC_c	ALC_n	ALC_c	ALC_n
Java	DNSJava	24-Mar-05	26-Apr-04	21-Jan-05	24-Apr-04	31-Mar-05	19-Apr-04
	Ant-Contrib	22-Sep-06	03-Aug-06	18-Sep-06	02-Aug-06	08-Sep-06	03-Aug-06
	Carol	25-Nov-07	18-Jan-07	25-Nov-07	14-Jan-07	12-Jun-05	27-Feb-07
	Plandora	31-Jan-11	01-Feb-11	01-Feb-11	01-Feb-11	31-Jan-11	01-Feb-11
C	Ctags	27-May-07	31-Dec-06	24-Mar-07	31-Dec-06	17-Sep-06	01-Jan-07
	Camellia	04-Nov-07	14-Nov-07	17-Jul-08	14-Nov-07	8-Feb-09	9-Nov-07
	QMail Admin	07-Nov-03	24-Oct-03	19-Nov-03	24-Oct-03	26-Nov-03	24-Oct-03
	Hash Kill	14-Jul-10	02-Dec-10	27-Jul-10	02-Dec-10	19-Jul-10	02-Dec-10
C#	GreenShot	11-Jun-10	21-Jun-10	12-Jun-10	21-Jun-10	20-Jun-10	20-Jun-10
	ImgSeqScan	19-Jan-11	14-Jan-11	17-Jan-11	14-Jan-11	19-Jan-11	14-Jan-11
	Capital Resource	13-Dec-08	12-Dec-08	11-Dec-08	12-Dec-08	10-Dec-08	12-Dec-08
	MonoOSC	08-Apr-09	21-Mar-09	05-Mar-09	21-Mar-09	21-Jan-09	22-Mar-09

Table 8: Overall Instabilities of Cloned ($OICR_c$) and Non-cloned ($OICR_n$) code by our proposed new method

		Type 1		Type 2		Type 3	
	Systems	$OICR_c$	$OICR_n$	$OICR_c$	$OICR_n$	$OICR_c$	$OICR_n$
Java	DNSJava	12.41	12.62	20.85	16.67	21.21	16.56
	Ant-Contrib	37.16	7.28	3.65	8.53	10.61	7.94
	Carol	6.06	8.64	5.31	8.79	12.41	8.00
	Plandora	10.5	3.72	4.99	3.83	6.17	3.36
C	Ctags	4.63	9.37	9.72	9.33	5.90	9.65
	Camellia	16.34	5.31	24.84	5.29	25.60	8.56
	QMailAdmin	49.84	32.37	44.77	32.38	52.50	30.72
	Hash Kill	7.42	53.69	0.71	53.44	11.75	55.23
C#	GreenShot	10.76	10.18	8.27	10.29	10.63	10.24
	ImgSeqScan	0	243.35	0	246.23	0	247.34
	CapitalResource	0	9.88	0	9.78	2.05	9.83
	MonoOSC	7.28	37.32	8.82	36.96	12.56	37.89

Table 9: Average Age in days of Cloned (AA_c) and Non-cloned (AA_n) code by the proposed variant

		Type 1		Type 2		Type 3	
	Systems	AA_c	AA_n	AA_c	AA_n	AA_c	AA_n
Java	DNSJava	2181	2441	2247	2443	2210.9	2446.9
	Ant-Contrib	853.6	903.7	896.1	903.3	870.6	904.4
	Carol	189.6	210.9	190.3	211.3	227	209.6
	Plandora	51.82	51.32	50.6	51.4	51.5	51.32
C	Ctags	1301.4	1345.2	1351.9	1345	1564.8	1343.4
	Camellia	1066.8	1056.7	810.9	1057.3	604.9	1062.4
	QMail Admin	2664.2	2678.1	2651.7	2678.2	2644.6	2678.3
	Hash Kill	261.5	118.5	250.3	118.4	257.9	118
C#	Green Shot	103.1	97.1	102.9	97.1	94.5	97.2
	ImgSeq Scan	14	20	15.6	20.3	14.4	20.4
	Capital Resource	86.7	86.5	88	86.5	89.3	86.5
	Mono OSC	315.4	313.5	347.9	313	378	312.3

Table 10: Comparative Stability Scenarios

	Methods / Systems	Krinke [13]			Hotta et al.[7]			Variant			Proposed new		
		T1	*T2*	*T3*	*T1*	*T2*	*T3*	*T1*	*T2*	*T3*	*T1*	*T2*	*T3*
Java	DNSJava	⊖	⊖	⊖	⊖	⊖	⊕	⊖	⊖	⊖	⊕	⊖	⊖
	Ant-Contrib	⊖	⊖	⊖	⊖	⊖	⊕	⊖	⊖	⊖	⊖	⊕	⊖
	Carol	⊖	⊖	⊕	⊖	⊖	⊖	⊖	⊖	⊕	⊕	⊕	⊖
	Plandora	⊕	⊕	⊕	⊕	⊕	⊕	⊕	⊖	⊕	⊖	⊖	⊖
C	Ctags	⊖	⊖	⊕	⊖	⊖	⊖	⊖	⊕	⊕	⊕	⊖	⊕
	Camellia	⊕	⊖	⊖	⊖	⊖	⊖	⊕	⊖	⊖	⊖	⊖	⊖
	QMailAdmin	⊖	⊖	⊖	⊖	⊖	⊖	⊖	⊖	⊖	⊖	⊖	⊖
	Hash Kill	⊕	⊕	⊕	⊕	⊕	⊕	⊕	⊕	⊕	⊕	⊕	⊕
C#	GreenShot	⊕	⊕	⊕	⊖	⊖	⊖	⊕	⊕	⊖	⊖	⊕	⊖
	ImgSeqScan	⊖	⊖	⊖	⊕	⊕	⊕	⊖	⊖	⊖	⊕	⊕	⊕
	CapitalResource	⊖	⊕	⊕	⊕	⊕	⊕	⊕	⊕	⊕	⊕	⊕	⊕
	MonoOSC	⊖	⊕	⊕	⊕	⊕	⊕	⊕	⊕	⊕	⊕	⊕	⊕

⊕=*Cloned Code More Stable*
⊖=*Non-Cloned Code More Stable*
T1, T2 and T3 denote clone types 1, 2 and 3 respectively

Table 11: Overall stability decisions by methods

Lang.	**Java**				**C**				**C#**			
Clone Types	DNSJava	Ant-Contrib	Carol	Plandora	Ctags	Camellia	QMail Admin	Hash Kill	GreenShot	ImgSeqScan	Capital Resource	MonoOSC
Type-1	⊖	⊖	⊖	⊕	⊖	⊗	⊖	⊕	⊗	⊗	⊕	⊕
Type-2	⊖	⊖	⊖	⊗	⊖	⊖	⊖	⊕	⊕	⊗	⊕	⊕
Type-3	⊖	⊖	⊗	⊕	⊕	⊖	⊖	⊕	⊖	⊗	⊕	⊕

⊕=*Most of the methods agree with* ⊕
⊖=*Most of the methods agree with* ⊖
⊗=*Decision conflict (Two methods agree with* ⊕ *and the remaining two methods agree with* ⊖*)*

For example, in Table 10 three methods (excluding our proposed new method) agree there is higher Type-1 clone instability in 'Ctags'. For the Type-3 case in 'Carol', two methods (the method proposed by Krinke and the proposed variant of Krinke's method) agree there is higher cloned code stability, whereas the other two methods agree there is higher non-cloned code stability. Thus, in Table 11, Type-1 clones for 'Ctags' is marked with '⊖' and Type-3 clones for 'Carol' is marked with ⊗.

6.1 Analysis Regarding the Comparative Stability of Cloned and Non-cloned Code

6.1.1 Analysis regarding UP and UPHL

From the graph in Fig.1 presenting the unstable proportions of cloned and non-cloned code we see that the unstable proportion of non-cloned code is always higher than that of cloned code. This is obvious because a larger code region would generally require more modifications to be maintained. To get more precise information we determined the UPHL for cloned and non-cloned regions of each of the candidate systems. The comparative scenario between $UPHL_c$ and $UPHL_n$ is presented in Fig. 2.

According to this graph, for most of the subject systems (11 out of 12) $UPHL_c > UPHL_n$. Thus, *cloned code is more likely to be modified compared to non-cloned code.* In answer to the first research question (RQ 1) we can say that *cloned code is modified more often than non-cloned code.*

We performed the MWW (Mann-Whitney Wilcoxon) test [17] on the observed values ($UPHL_c$ and $UPHL_n$) for the 11 subject systems with a higher $UPHL_c$ to determine whether $UPHL_c$ is significantly higher than $UPHL_n$. The two tailed probability value (p-value) for this test is 0.00244672. The p-value is much less than 0.05 and it implies that $UPHL_c$ is significantly higher than $UPHL_n$ for our candidate subject systems. Thus, according to our experimental result, *cloned code is modified significantly more often than non-cloned code.*

6.1.2 Analysis regarding PCRM

The graph in Fig. 3 presents the comparison between the $PCRM_c$ and $PCRM_n$ of the each of the candidate systems. We see that for most of the subject systems (10 out of 12), $PCRM_c > PCRM_n$. For one (ImgSeqScan) of the remaining two subject systems, $PCRM_c = 0$ because the cloned regions of this subject system did not have any modifications. Thus, in answer to the second research question (RQ 2) we can say that *the proportion of cloned regions (i.e., the proportion of cloned LOC) modified due to effective commits is generally higher than the proportion of the non-cloned regions getting modified due to effective commits.*

Considering the 10 subject systems with higher $PCRM_c$ we performed the MWW (Mann-Whitney Wilcoxon) test [17] on the observed values of $PCRM_c$ and $PCRM_n$ to determine whether $PCRM_c$ is significantly higher than $PCRM_n$ for these systems. The two tailed probability value (p-value) regarding the test is 0.01854338. We see that the p-value is less than 0.05. Thus, the difference between $PCRM_c$ and $PCRM_n$ is marginally significant according to our findings.

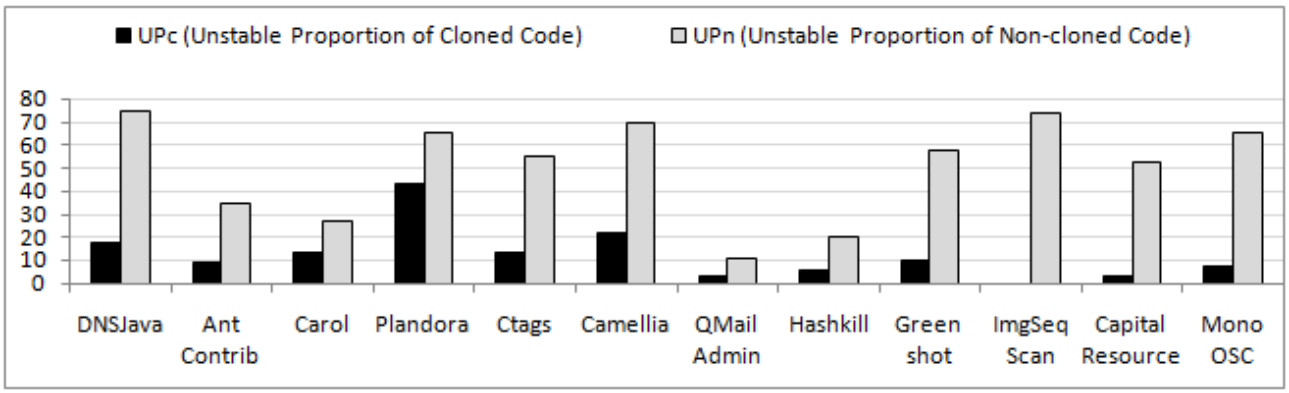


Figure 1: Unstable proportions (UP) of cloned and non-cloned code

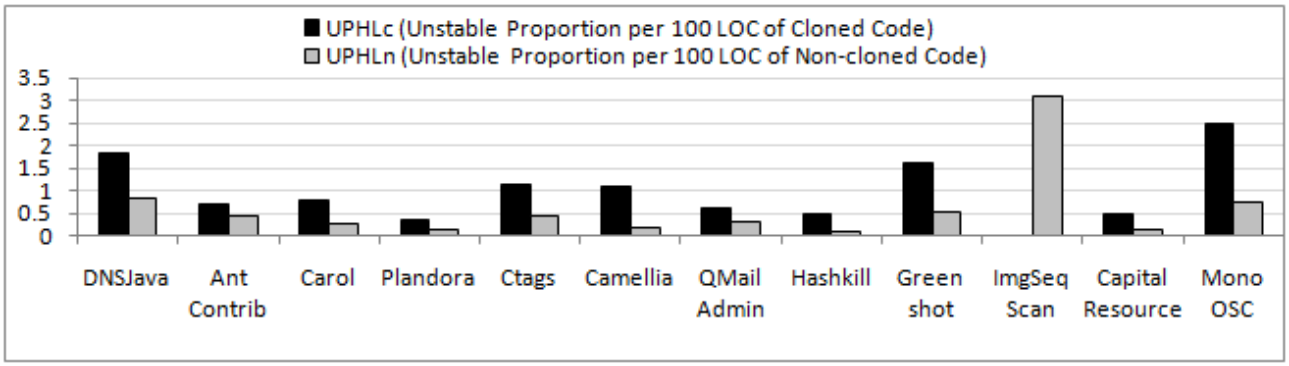


Figure 2: Unstable proportion per 100 LOC (UPHL) of cloned and non-cloned code

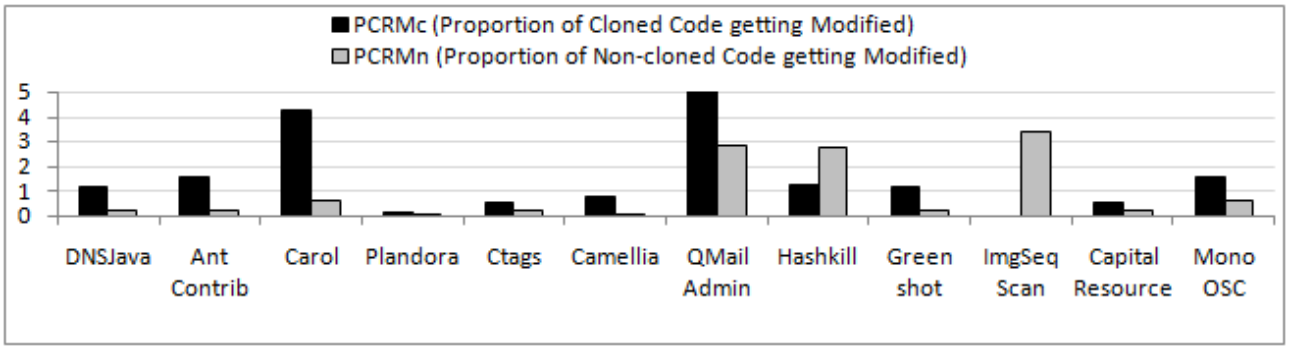


Figure 3: Proportions of cloned and non-cloned regions getting modified (PCRM)

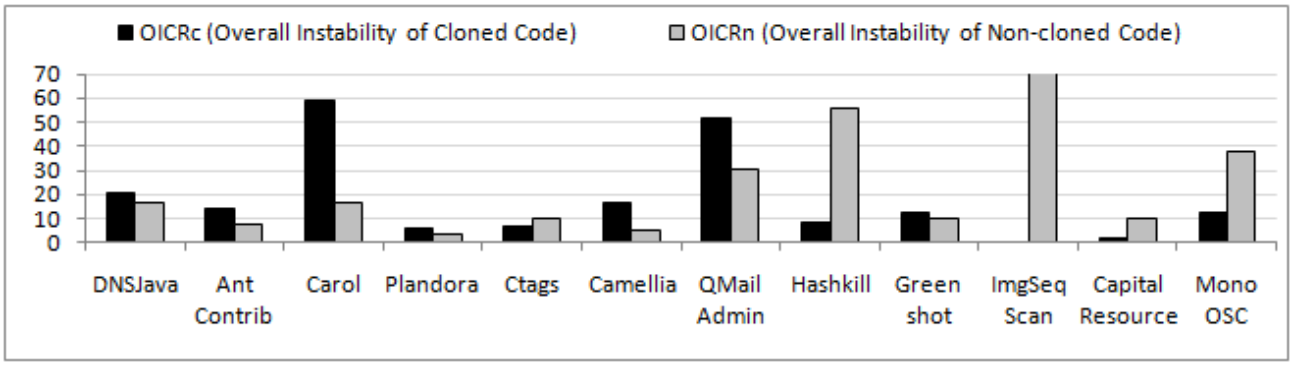


Figure 4: Overall instabilities of cloned and non-cloned code (OICR)

6.1.3 Analysis regarding OICR

The comparison of the overall instabilities of the cloned and non-cloned regions of the candidate systems is presented in Fig. 4. According to this graph seven subject systems have higher overall instability of cloned code (higher $OICR_c$) while the remaining five subject systems have the opposite. For one (ImgSeqScan) of these remaining five systems, $OICR_c$ equals zero because cloned regions of this system did not get modified. In other words, the cloned regions of this system did not have any effective commits. However, the comparison scenario presented by the graph indicates that *in the presence of effective commits, overall instability of a cloned region is generally higher than that of a non-cloned region.*

6.1.4 Analysis Result

From the analysis of our experimental results regarding the comparative stability of cloned and non-cloned code, we see that (1) cloned code gets modified significantly more often than non-cloned code, (2) the proportion of cloned region getting modified due to effective commits (defined in the last paragraph of Section 3.5) is higher than that of non-cloned region, and (3) finally, in answer to the third research question (RQ 3) we can say that the overall instability of cloned code is generally higher than that of non-cloned code. The comparative scenario implies that software clones are generally less stable than non-cloned code and thus, clones should be managed with proper care to increase the stability of software systems.

6.2 Type Centric Analysis

In this analysis, we tried to answer the fourth research question (Table 1) by investigating how a particular method's decisions on a particular subject system vary with the variation of clone types. We have the following observations.

The stability decisions made by a method on a specific subject system corresponding to three types of clones are similar for 31 cases with some minor variations for the remaining cases. That is, Table 10, shows there are 64.58% similar cases among 48 cases. Each case consists of three decisions for three types of clones made by a particular method on a particular subject system. As an example of variations, consider the decisions made by Hotta's method for 'DNSJava'. For the Type-3 case (Table 7), $MF_d < MF_n$ suggests that Type-3 clones are more stable than the corresponding non-cloned code. However, according to this method, Type-1 and Type-2 clones of 'DNSJava' are much less stable than non-cloned code (the difference of MFs for the Type-3 case is smaller compared to the differences for the other two cases). We analyzed the experimental results in the following two ways.

6.2.1 Analysis 1

This analysis is based on the agreement-disagreement scenario of the Table 11. According to the agreed decisions points (Table 11) of the Type-1 case:

(i) clones decrease the stability of a system with probability = No. of cells with cloned code less stable/total no. of cells = 5/12 = 0.42.

(ii) non-cloned code decreases the stability of a system with probability = 4/12 = 0.33.

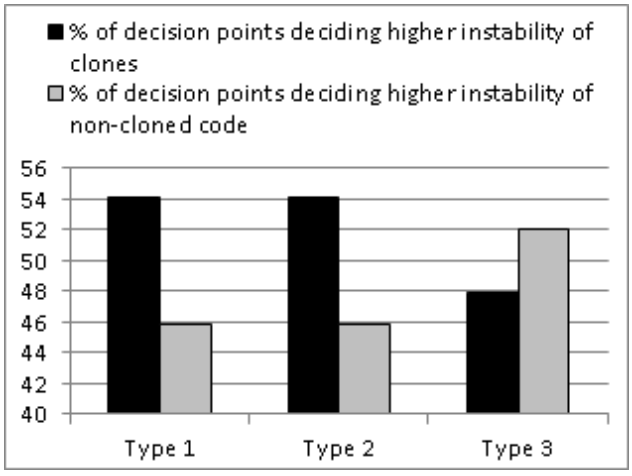


Figure 5: Type centric analysis

For the Type-2 case, these two probabilities are 0.50 (for cloned code) and 0.33 (for non-cloned code) respectively. So, for both of these cases (Type-1 and Type-2) cloned code has a higher probability of decreasing the system's stability. But, for Type-3 case these two probabilities are the same (0.42 for both cloned and non-cloned code). We see that for both Type-1 and Type-2 cases, clones have higher probability of making a system unstable compared to the probability of corresponding non-cloned code. However, Type-3 clones do not appear to be more unstable than non-cloned code according to our findings.

6.2.2 Analysis 2

In this case, we analyzed the data in Table 10. In this table, each type of clones contribute 48 decision points in total. Considering these 48 decision points (for each type of clones) we calculated the proportions of decision points agreeing with higher instability of cloned or non-cloned code. These proportions are plotted in Fig. 5.

According to this graph, the higher proportion of decision points belonging to both Type-1 and Type-2 case agree with the higher instability of cloned code compared to the Type-3 case. Thus, Type-1 and Type-2 clones are more vulnerable in the software systems compared to the vulnerability exhibited by Type-3 clones.

6.2.3 Analysis Result

Both Type-1 clones (created by exact copy-paste activities), and Type-2 clones (created by renaming identifiers and changing data types) should be given more attention during the development and maintenance phase.

6.3 Method Centric Analysis

We see that Table 10 contains 144 decision points where each method contributes 36 decisions (corresponding to 12 systems and 3 clone types). From this we can retrieve the decision making scenario presented in Table 12 exhibited by the candidate methods. According to this table, the majority of the methods (three out of four) agree there is higher instability in cloned code. However, there are decision disagreements among the methods. The disagreements have been analyzed in the following way.

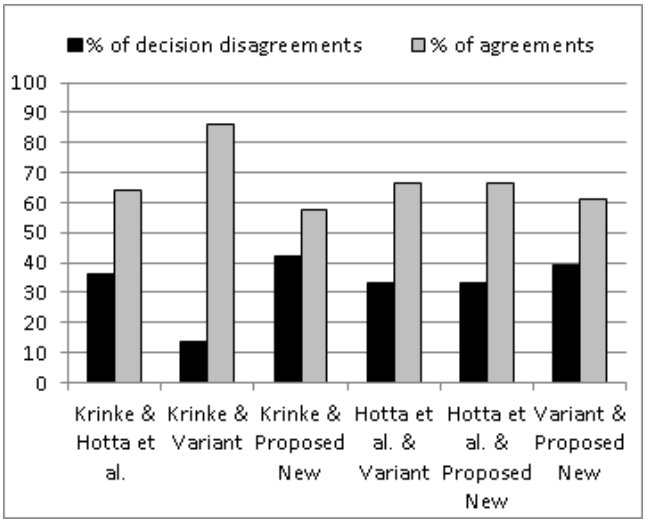


Figure 6: Method centric analysis regarding disagreements

6.3.1 Analysis of disagreements

We see that in Table 10 each method contributes 36 decision points. From this table we determined the percentage of disagreements for each pair of methods. For each decision point belonging to a particular method we get a corresponding decision point in another method. For each pair of methods, we see whether the corresponding decision points are conflicting (making different stability decisions) or not. From 36 sets of corresponding points for a particular method pair, we determine the proportion of conflicting sets. These proportions are shown in the graph of Fig.6. We have the following observations from the graph.

(1) We see that the method proposed by Krinke and its variant have the lowest proportion of conflicts. As both of these methods work on the last revision of a subject system they should exhibit a higher proportion of agreements in their decisions. The reason behind the observed disagreements is that the proposed variant excludes blank lines and comments from consideration while calculating average ages of cloned and non-cloned code. But, Krinke's method is biased by the blank lines and comments because it does not exclude these from consideration while determining average last change dates.

(2) Each of the methods proposed by Krinke and its variant has strong disagreements with each of the other two methods (proposed by Hotta et al. and our proposed new method). The reason behind this disagreement is that while both of the methods proposed by Hotta et al. and our proposed new method examine each of the existing revisions of a particular software system, the other two methods examine only the last revision for making stability decisions. In the following example we explain an observed strong disagreement.

Example and explanation of a strong disagreement: We consider 'ImqSeqScan' as an (extreme) example. For each clone type, each of the methods proposed by Hotta et al. and the proposed new method shows strong disagreement to the decision of Krinke's method and its variant. Each of the three types of clones was suggested to be more stable than non-cloned code by the methods proposed by Hotta et al. (Table 7) and the proposed new method (Table 8). However, both Krinke's method and its variant yield the opposite decisions (Table 5 and 9). More interestingly, both Hotta et al.'s method and our proposed new method reveal that the cloned regions of 'ImgSeqScan' did not receive any change (modification frequencies of cloned code is 0 according to the Table 7, overall instability of cloned code is 0 according to the Table 8) during the entire lifetime (consisting of 73 commit transactions) where the other two methods show that the cloned code is significantly younger. In this case the regions of cloned code have only been created lately and have not been modified after creation. The following explanation will clarify this.

Table 12: Stability w.r.t. candidate methods

Decision Parameters	% of Decision Points			
	Krinke [13]	Hotta et al.[7]	Proposed variant	Proposed New
Non-cloned code more stable (cloned code less stable)	55.56	52.78	52.78	47.22
Cloned code more stable	44.44	47.22	47.22	52.78

Suppose a subject system receives 100 commit transactions. Some clone fragments were created in some of these commits but no existing clone fragment was modified at all. In such a case, both Hotta et al.'s method and our proposed new method will see that there are no modifications in the cloned region. As a result, MF_d (for Hotta et al.'s method) and $OICR_c$ (for the proposed new method) will be zero. On the other hand, the *blame* command will retrieve the creation dates of the clone fragments existing in the last revision of the system and Krinke's method will determine the average last change date for the cloned region considering these creation dates. If the creation dates of some clone fragments are newer than the modification dates of non-cloned fragments which forces the average last change date of the cloned region to be newer than that of the non-cloned region, Krinke's method will suggest cloned code to be less stable than non-cloned code. Thus, the cloned or non-cloned region of a subject system might be represented to be less stable than its counterpart even if it does not undergo any modifications during the entire evolution time while its counterpart does.

(3) The proposed new method disagrees with Hotta et al.'s method for 33.33% cases. The main reason behind this disagreement has already been explained in Section 3.6. Plandora is an extreme example of such disagreement. According to Hotta et al.'s method, each type of clones of this subject system are more stable than the corresponding non-cloned code. But, our proposed new method makes the opposite decision in each case.

Finally, in answer to the fifth research question (RQ 5) we can say that the stability decisions made by the candidate methods are often not similar and both Hotta et al.'s method and our proposed new method have strong disagreements with the other two methods in many cases.

Table 13: Stability w.r.t. programming languages

Decision Parameters	% of Agreed Decision Points		
	Java	C	C#
Non cloned code more stable (Cloned code less stable)	66.67	58.33	8.33
Cloned code more stable	16.67	33.33	58.33
Conflicting decisions	16.66	8.34	33.34

Table 14: Fisher's Exact Tests for prog. languages

	Java	C	Java	C#	C	C#
CCLS	67	58	67	8	58	8
NCLS	17	33	17	58	33	58
DC	17	8	17	33	8	33
P = 0.0103			P <0.0001		P <0.0001	

CCLS = Cloned Code Less Stable
NCLS = Non-cloned Code Less Stable
DD = Decision Conflicts

6.3.2 Analysis result

Considering all candidate methods and metrics we see that cloned code (all three types) has a higher probability to force a system into an unstable state compared to non-cloned code. According to Table 10, cloned code is less stable than non-cloned code for 75 cells (among 144 cells). The opposite is true for the remaining 69 cells. So the probability by which cloned code makes the system unstable is 75/144 = 0.52 which outweighs the probability of non-cloned code (0.48). Though the difference between the probabilities is very small, it disagrees with the conclusion drawn by both Krinke [13] and Hotta et al.[7] regarding comparative stability. Thus, clones should be carefully maintained and refactored (if possible) instead of keeping aside.

6.4 Language Centric Analysis

We performed language centric analysis in two ways.

6.4.1 Analysis 1

This analysis is based on the agreement-disagreement scenario of Table 11. Our set of subject systems consists of four systems from each of the three languages (Java, C and C#). In Table 11, each language contributes 12 (4 subject systems, 3 clone types) decision points. Considering these decision points we retrieved the language specific stability scenario presented in Table 13.

According to this table, both Java and C exhibit higher cloned code instability: 66.67% and 58.33% of the cases, respectively. The majority of the candidate methods agreed there is higher instability in cloned code. An exactly the opposite scenario was observed for C#. Moreover, for C# we can observe the highest proportion (33.33%) of decision conflicts. Thus, clones in both Java and C systems have a higher probability of making a system unstable compared to the clones in C# systems. Our Fisher's Exact Test results regarding the language centric statistics are described below.

Fisher's Exact Test: We performed Fisher's exact tests [5] on the three possible paired-combinations of the three languages using the values in Table 13 to see whether there are significant differences among the observed proportions of different languages. We defined the following null hypothesis. The values in Table 13 were rounded before using in Fisher's exact test.

Null Hypothesis: There is no significant difference between the stability scenarios presented by different programming languages.

From Table 14 we see the *P value* for each paired combination of programming languages is less than 0.05. This rejects the null hypothesis and confirms that there are significant differences among the observed scenarios of different programming languages.

6.4.2 Analysis 2

This analysis is based on the Table 10. In this table we see that each method contributes 12 decision points for each programming language. For each combination of method and language we determined two proportions: (1) the proportion of decision points agreeing there is higher cloned code instability, and (2) the proportion of decision points agreeing there is higher non-cloned code instability. These proportions are presented in Fig. 7.

In the bar chart (Fig. 7) we see that for each method, a higher proportion of decision points belonging to both Java and C agree there is higher cloned code instability. The opposite scenario is exhibited by C#. For each method, a higher proportion of decision points (belonging to C#) agree that there is higher cloned code stability. Thus, from this analysis we can also say that clones in both Java and C systems have a higher probability of making a system unstable compared to the clones in C#.

6.4.3 Analysis result

In answer to the sixth research question (RQ 6) we can say that clones in both Java and C systems exhibit significantly higher instability compared to the clones in C# systems and so developers as well as project managers should be more careful regarding clones during software development using these languages (Java and C).

6.5 System Centric Analysis

In the system centric analysis we investigated whether system sizes and system ages affect the comparative stabilities. In this investigation we wanted to observe how modifications occur in the cloned and non-cloned code of a subject system as the system becomes older and bigger in size. So, we recorded and plotted the modification frequencies of four subject systems for different revisions. We chose 'DNSJava', 'Carol', 'MonoOSC' and 'Hashkill' in this investigation. 'DNSJava' and 'Carol' have a large number of revisions compared to the revision numbers of other two systems. On the other hand 'Hashkill' is much bigger than the remaining three systems in terms of LOC. So, selecting these system we have a range of systems in terms of LOCs and revision numbers covering three languages. Also, these subject systems yielded contradictory stability scenarios for the method proposed by Hotta et al.

We present four line graphs (Fig. 8, Fig. 9, Fig. 10, Fig. 11) for these subject systems plotting their modification fre-

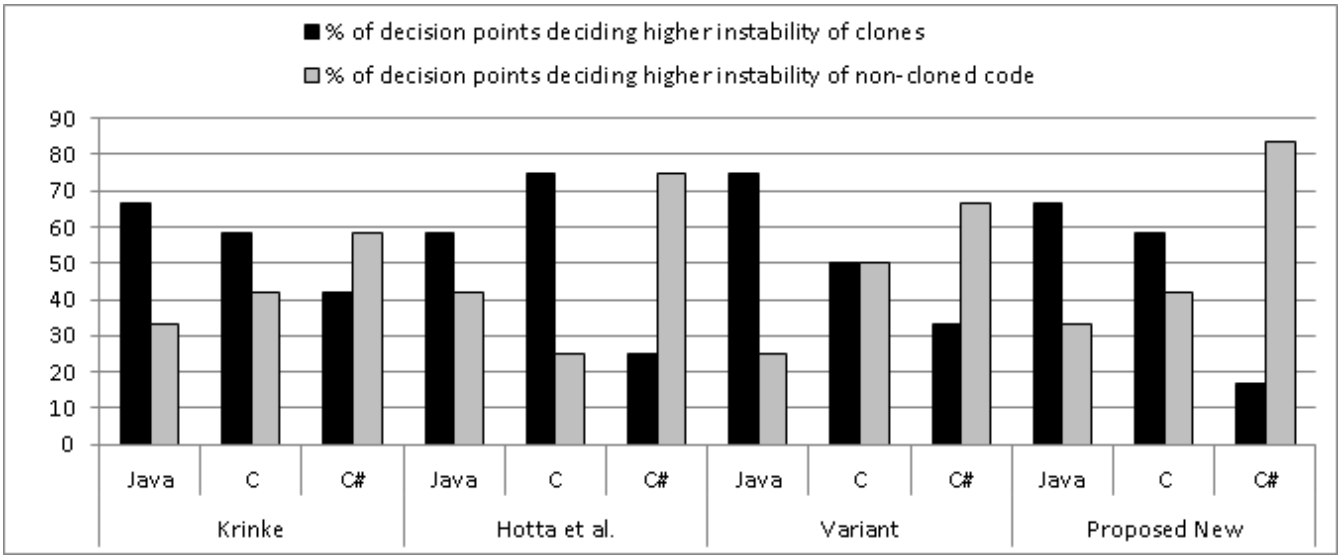


Figure 7: Language centric analysis

quencies for each of the revisions beginning with the second revision. For each revision r ($r \geq 2$), the modification frequencies of cloned and non-cloned code plotted in the graph were calculated by considering all revisions from 1 to r. The intention is to calculate the modification frequencies for r ($r \geq 2$) considering r as the current last revision. These visual representations of the gradual changes of modification frequencies reflect the exact trends of how the cloned and non-cloned regions were modified in the development phase.

In both Fig. 8 and Fig. 10, we see that for some development time cloned code was more stable than non-cloned code and vice versa. But the graph in Fig. 9 shows that for most of the development time, cloned code was modified more frequently than the non-cloned code. The graph in Fig. 11 exhibits a completely different scenario. The four graphs exhibit no change consistency or bias. Also, in the case of Carol (Fig. 9) we see that although for most of the life time the modification frequency curves showed opposite characteristics, the curves tend to meet each other at the end. On the other hand, the curves of the other three systems seem to diverge from each other. From Table 3 we can see that these four systems are of diverse sizes and nature. Thus, the convergence or divergence of the modification frequency curves is not dependent on the system sizes. So, RQ 7 can be answered by the observation that system sizes and system ages do not affect the stability of cloned and non-cloned code in a consistent or correlated way.

It is worth noting that every system can have a different development strategy which can affect changes to cloned and non-cloned code. For example, programmers might be afraid of changing cloned code because of the risk of inconsistent changes and would try to restrict the changes to the non-cloned code. Another possibility is that developers are advised to not change any code of other authors and thus are forced to create a clone in order to apply a change. However, such development strategies cannot be identified by looking at the change history alone and thus it is not possible to measure the impact on cloned and non-cloned code.

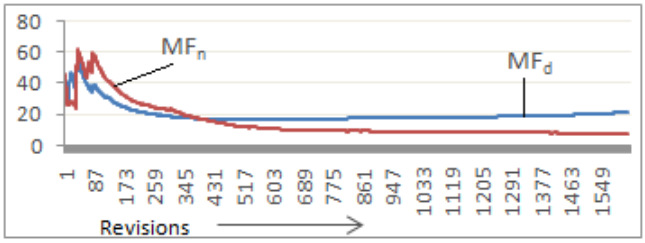


Figure 8: MFs for DNSJava (Type-1 case. Non-cloned code is more stable)

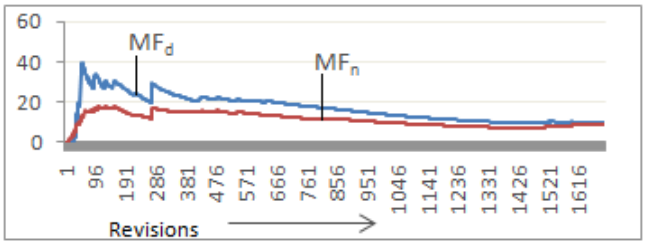


Figure 9: MFs for Carol (Type-3 case. Non-cloned code is more stable)

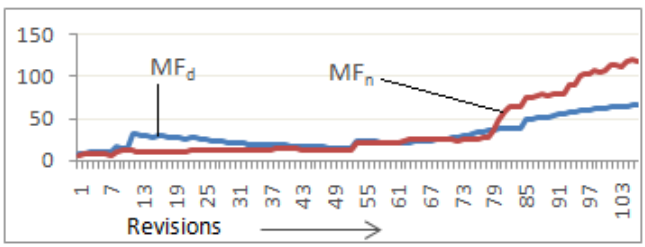


Figure 10: MFs for Hash Kill (Type-3 case. Cloned code is more stable)

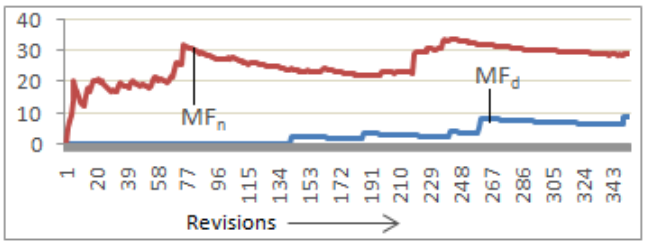


Figure 11: MFs for MonoOSC (Type-1 case. Cloned code is more stable)

7. THREATS TO VALIDITY

In the experimental setup section we mentioned the clone granularity level (block clones), difference thresholds and identifier renaming options that we have used for detecting three clone types. Different setups in corresponding clone types might result in different stability scenarios. However, the NiCad setups that we have used for detecting three types clones are considered standard [23, 24, 26, 4, 29] and thus we contend that the clone detection results that we have investigated are reliable.

8. CONCLUSION

In this paper we presented an in-depth investigation on the comparative stabilities of cloned and non-cloned code. We presented a five dimensional analysis of our experimental results to answer seven research questions. The ultimate aim of our investigation is to find out the changeabilities exhibited by different types of clones and languages and whether there is any yet-undiscovered consistency in code modification biasing the stability scenarios. We introduced a new method that calculates four new stability related metrics for the purpose of analysis.

According to our comparative stability centric analysis, cloned code is more unstable (as well as vulnerable) than non-cloned code because clones get modified significantly more often than non-cloned code (supported with Mann-Whitney-Wilcoxon tests). Also, the proportion of the cloned regions modified in effective commits is significantly higher than the proportion of non-cloned regions being modified.

However, our system centric analysis suggests that there are no existing biases in the modifications as well as stabilities of cloned and non-cloned code, and system development strategy can play an important role in driving comparative stability scenarios.

Our type centric analysis reveals that Type-1 (exact clones) and Type-2 (clones with differences in identifier names and data types) clones are possibly harmful for a system's stability. They exhibit higher probabilities of instabilities than the corresponding non-cloned code. Thus, these clone types should be given more attention both from a development and a management perspective.

Our language centric analysis discovers that clones of Java and C systems show higher modification probabilities compared to those of C# systems. This argument is also supported by statistical proof using Fisher's exact test (2 tailed).

Our method centric analysis discovers the causes of strong and weak disagreements of the candidate methodologies in making stability decisions. In this analysis we evaluated 144 decision points of comparative stabilities and found that cloned code exhibits higher changeability than that of non-cloned code which contradicts the already established bias ([13, 7]) regarding comparative stabilities of cloned vs. non-cloned code. Thus, cloned code is not necessarily stable as was observed in the previous studies [7, 13] and clones should be managed.

Our future plan is to perform an exhaustive empirical study for further analysis of the impacts of clones using several clone detection tools, methods and a wider range of subject systems.

Acknowledgments: This work is supported in part by the Natural Science and Engineering Research Council of Canada (NSERC).

9. REFERENCES

[1] Aversano, L., Cerulo, L., and Penta, M. D., "How clones are maintained: An empirical study", in Proc. *The 11th European Conference on Software Maintenance and Reengineering (CSMR)*, 2007, pp. 81-90.

[2] CCFinderX. `http://www.ccfinder.net/ccfinderxos.html`

[3] Cordy, J. R., and Roy, C. K., "The NiCad Clone Detector", in Proc. *The Tool Demo Track of the 19th International Conference on Program Comprehension (ICPC)*, 2011, pp. 219-220.

[4] Cordy, J. R., and Roy, C. K., "Tuning Research Tools for Scalability and Performance: The NICAD Experience", in *Science of Computer Programming*, 2012, 26 pp. (to appear)

[5] Fisher's Exact Test. `http://in-silico.net/statistics/fisher_exact_test/2x3`.

[6] Göde, N., and Harder, J., "Clone Stability", in Proc. *The 15th European Conference on Software Maintenance and Reengineering (CSMR)*, 2011, pp. 65-74.

[7] Hotta, K., Sano, Y., Higo, Y., and Kusumoto, S., "Is Duplicate Code More Frequently Modified than Non-duplicate Code in Software Evolution?: An Empirical Study on Open Source Software", in Proc. *The Joint ERCIM Workshop on Software Evolution (EVOL) and International Workshop on Principles of Software Evolution (IWPSE)* , 2010, pp. 73-82

[8] Juergens, E., Deissenboeck, F., Hummel, B., and Wagner, S., "Do Code Clones Matter?", in Proc. *The 31st International Conference on Software Engineering (ICSE)*, 2009, pp. 485-495.

[9] Kapser, C., and Godfrey, M. W., ""Cloning considered harmful" considered harmful: patterns of cloning in software", in *Journal of Empirical Software Engineering*. 13(6), 2008, pp. 645-692.

[10] Kim, M, Sazawal, V., Notkin, D., and Murphy, G. C., "An empirical study of code clone genealogies", in Proc. *The joint meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering (ESEC-FSE)*, 2005, pp. 187-196.

[11] Krinke, J., "A study of consistent and inconsistent changes to code clones", in Proc. *The 14th Working*

Conference on Reverse Engineering (WCRE), 2007, pp. 170-178.

[12] Krinke, J., "Is cloned code more stable than non-cloned code?", in Proc. *The 8th IEEE International Working Conference on Source Code Analysis and Manipulation (SCAM)*, 2008, pp. 57-66.

[13] Krinke, J., "Is Cloned Code older than Non-Cloned Code?", in Proc. *The 5th International Workshop on Software Clones (IWSC)*, 2011, pp.28-33.

[14] Lozano, A., Wermelinger, M., and Nuseibeh, B., "Evaluating the Harmfulness of Cloning: A Change Based Experiment", in Proc. *The 4th International Workshop on Mining Software Repositories (MSR)*, 2007, pp. 18-21.

[15] Lozano, A., and Wermelinger, M., "Tracking clones' imprint", in Proc. *The 4th International Workshop on Software Clones (IWSC)*, 2010, pp. 65-72.

[16] Lozano, A., and Wermelinger, M., "Assessing the effect of clones on changeability", in Proc. *The 24th IEEE International Conference on Software Maintenance (ICSM)*, 2008, pp. 227-236.

[17] Mann-Whitney-Wilcoxon Test: http://elegans.som.vcu.edu/ leon/stats/utest.html

[18] Mondal, M., Roy, C. K., Rahman, M. S., Saha, R. K., Krinke, J., and Schneider, K. A., "Comparative Stability of Cloned and Non-cloned Code: An Empirical Study", in Proc. *The 27th Annual ACM Symposium on Applied Computing (SAC)*, 2012, pp. 1227–1234.

[19] Mondal, M., Roy, C. K., and Schneider, K. A., "Dispersion of Changes in Cloned and Non-cloned Code", in Proc. *The 6th International Workshop on Software Clones (IWSC)*, 2012, pp. 29-35 .

[20] Roy, C. K., and Cordy, J. R., "A mutation / injection-based automatic framework for evaluating code clone detection tools", in Proc. *The IEEE International Conference on Software Testing, Verification, and Validation Workshops* , 2009, pp. 157-166.

[21] Roy, C. K., and Cordy, J. R., "NICAD: Accurate Detection of Near-Miss Intentional Clones Using Flexible Pretty-Printing and Code Normalization" in Proc. *The 16th IEEE International Conference on Program Comprehension (ICPC)*, 2008, pp. 172-181.

[22] Roy, C. K., Cordy, J. R., and Koschke, R., "Comparison and Evaluation of Code Clone Detection Techniques and Tools: A Qualitative Approach", in *Science of Computer Programming*, 74 (2009) 470-495, 2009.

[23] Roy, C. K., and Cordy, J. R., "Near-miss Function Clones in Open Source Software: An Empirical Study", in *Journal of Software Maintenance and Evolution: Research and Practice*, 22(3), 2010, pp. 165-189.

[24] Roy, C. K., and Cordy, J. R., "An Empirical Evaluation of Function Clones in Open Source Software", in Proc. *The 15th Working Conference on Reverse Engineering (WCRE)*, 2008, pp. 81-90.

[25] Roy, C. K., and Cordy, J. R., "Scenario-based Comparison of Clone Detection Techniques", in Proc. *The 16th IEEE International Conference on Program Comprehension (ICPC)*, 2008, pp.153-162.

[26] Saha, R. K., Roy, C. K., and Schneider, K. A., "An Automatic Framework for Extracting and Classifying Near-Miss Clone Genealogies", in Proc. *The 27th IEEE International Conference on Software Maintenance (ICSM)*, 2011, pp. 293-302.

[27] Saha, R. K., Asaduzzaman, M., Zibran, M. F., Roy, C. K., and Schneider, K. A., "Evaluating code clone genealogies at release level: An empirical study", in Proc. *The 10th IEEE International Conference on Source Code Analysis and Manipulation (SCAM)*, 2010, pp. 87-96.

[28] Thummalapenta, S., Cerulo, L., Aversano, L., and Penta, M. D., "An empirical study on the maintenance of source code clones", in *Journal of Empirical Software Engineering (ESE)*, 15(1), 2009, pp. 1-34.

[29] Zibran, M. F., Saha, R. K., Asaduzzaman, M., and Roy, C. K., "Analyzing and Forecasting Near-miss Clones in Evolving Software: An Empirical Study", in Proc. *The 16th IEEE International Conference on Engineering of Complex Computer Systems (ICECCS)*, 2011, pp. 295-304.

ABOUT THE AUTHORS:

Manishankar Mondal is a graduate student in the Department of Computer Science of the University of Saskatchewan, Canada under the supervision of Dr. Chanchal Roy and Dr. Kevin Schneider. He is a lecturer at Khulna University, Bangladesh and currently on leave for pursuing his higher studies. He received the Best Paper Award from the 27th Symposium On Applied Computing (ACM SAC 2012) in the Software Engineering Track. His research interests are software maintenance and evolution including clone detection and analysis, program analysis, empirical software engineering and mining software engineering.

Chanchal Roy is an assistant professor of Software Engineering/Computer Science at the University of Saskatchewan, Canada. While he has been working on a broad range of topics in Computer Science, his chief research interest is Software Engineering. In particular, he is interested in software maintenance and evolution, including clone detection and analysis, program analysis, reverse engineering, empirical software engineering and mining software repositories. He served or has been serving in the organizing and/or program committee of major software engineering conferences (e.g., ICSM, WCRE, ICPC, SCAM, ICSE-tool, CASCON, and IWSC). He has been a reviewer of major Computer Science journals including IEEE Transactions on Software Engineering, International Journal of Software Maintenance and Evolution, Science of Computer Programming, Journal of Information and Software Technology and so on. He received his Ph.D. at Queen's University, advised by James R. Cordy, in August 2009.

Kevin Schneider is a Professor of Computer Science, Special Advisor ICT Research and Director of the Software Engineering Lab at the University of Saskatchewan. Dr. Schneider has previously been Department Head (Computer Science), Vice-Dean (Science) and Acting Chief Information Officer and Associate Vice-President Information and Communications Technology. Before joining the University of Saskatchewan, Dr. Schneider was CEO and President of Legasys Corp., a software research and development company specializing in design recovery and automated software engineering. His research investigates models, notations and techniques that are designed to assist software project teams develop and evolve large, interactive and usable systems. He is particularly interested in approaches that encourage team creativity and collaboration.

XFormsDB: An Extensible Web Application Framework Built upon Declarative W3C Standards

Markku Laine, Denis Shestakov, Petri Vuorimaa

Department of Media Technology, Aalto University
P.O. Box 15500, FI-00076 Aalto, Finland

{markku.laine, denis.shestakov, petri.vuorimaa}@aalto.fi

ABSTRACT

Most Web applications are based on a conventional three-tier architecture, in which the presentation, application logic, and data management are developed and maintained in separate tiers. The main disadvantage of this architecture is that it requires expertise in multiple programming languages, programming paradigms, and data models used in each tier. A single expert rarely masters all the technologies and concepts involved. In this paper, we introduce a *tier-expanding* architectural approach that unifies the client-side (presentation tier) and server-side (logic and data tiers) programming under a single model. We base our approach on a W3C-standardized client-side markup language, XForms, and its server-side extension proposed in this paper. We derive the extension requirements from the literature and use cases, and demonstrate their functionality on the example of a blog Web application. We also show how the extension can be implemented as part of a comprehensive Web application framework called *XFormsDB*. The XFormsDB framework is an extensible Web application framework built upon declarative W3C standards. It has four major advantages: (1) one programming language, (2) one data model, (3) based on W3C-standardized declarative markup, and (4) extensibility in all tiers. Our conclusion is that expanding the presentation tier to cover both application logic and data management functionality makes both the development and maintenance of small- and medium-sized Web applications easier.[1]

Categories and Subject Descriptors

D.2.11 [**Software Engineering**]: Software Architectures – *Languages*. D.3.3 [**Programming Languages**]: Language Constructs and Features – *Frameworks*. I7.2 [**Document and Text Processing**]: Document Preparation – *Markup Languages*.

General Terms

Design, Languages.

Keywords

Web Framework, Web Application, Web Development, Declarative Language, XForms.

[1] This work is based on an earlier work: SAC '12 Proceedings of the 2012 ACM Symposium on Applied Computing, Copyright 2012 ACM 978-1-4503-0857-1/12/03. http://doi.acm.org/10.1145/2245276.2245407.

1. INTRODUCTION

The constantly evolving Internet has grown from an information dissemination platform to the medium running variety of applications and services. Highly interactive data-driven Web applications—commonly known as Rich Internet Applications (RIAs) [9]—are now an integral part of our lives: we use them to pay our bills, work collaboratively with our colleagues, check weather conditions, play games, browse friends' photos, and blog about our experiences.

While the widespread adoption of RIAs has significantly improved the utility and user experience of the Web, developing such applications has dramatically increased in complexity. We can see this trend by examining Web applications based on a conventional three-tier Web application architecture [1]. In those applications, the structure and layout of a user interface is typically authored in HTML and CSS, whereas JavaScript handles the interaction. The server-side application logic, on the other hand, is implemented using an object-oriented or scripting language, such as Java, Ruby, or PHP. The client-server communication is handled using the HTML, XML, or JSON formats and asynchronous submissions, and the application data is managed with SQL statements. In addition, data-mapping libraries may be used for translating data from one format to another when moving data between different tiers. In this way, RIA developers not only need to know a multitude of systems, frameworks, best practices and languages, but also to deal with their conceptual dissimilarities [29]—indeed, the same Web application often consists of components written in imperative (e.g., Java and JavaScript) and declarative (e.g., CSS, HTML, and SQL) languages.

Unifying the client-side (presentation tier) and server-side (logic and data tiers) programming under a single model can simplify the Web application development and particularly reduce the skill set required from a developer. Reducing the number of technologies involved also makes an application more secure, as in general each technology is one more compromise in the overall application security. Generally, a unified model can be based on either server-side or client-side concepts. For instance, *Google Web Toolkit (GWT)*[2] realizes a server-side approach, in which a general-purpose programming language—namely, object-oriented imperative Java—is used to author the application logic both on the client and the server. GWT also allows authoring the Web application user interface portion in Java. However, because the user interface design and implementation almost always require

[2] Google Web Toolkit, http://developers.google.com/web-toolkit/

human involvement and judgment, while most of the server-side application logic can be covered by generic components, an approach based on a client-side programming language is a more compelling alternative.

Since our primary target group is users who are involved in Web content authoring and possess no or little programming skills, a declarative markup language is a proper candidate for this client-side based Web programming model. Thus, we chose XForms [4], an XML-based Web user interface language standardized by W3C. XForms addresses the most common problems found in HTML forms (e.g., dependency on imperative scripting languages, such as JavaScript) and eases dynamic Web form authoring by using declarative markup.

In this paper, we propose a *tier-expanding* architectural approach, which allows using the declarative XForms markup language on all three tiers of a Web application. We define the requirements for extending the language with common server-side and database-related functionality and present the design of the extension. We also introduce *XFormsDB*, a comprehensive Web application framework that supports the XForms markup language and its server-side extension proposed in this paper. Furthermore, we describe the architecture of the XFormsDB framework, present its implementation in detail, and show how it leverages declarative W3C standards on different tiers. Finally, we argue that the framework could significantly simplify the development and maintenance work of small- and medium-sized Web applications as well as reduce the skill set required from a developer. We developed the XFormsDB framework as an open source project and made it available under the MIT license. The presentation of the framework can be found elsewhere [22].

The rest of the paper is organized as follows. The next section reviews the literature relevant to this research. Section 3 provides the fundamentals of XForms and describes how the technology fits into the conventional three-tier Web application architecture. Then, an overview of XQuery is given in Section 4. In Sections 5 and 6, we present our approach of extending a client-side Web programming language with server-side functionality along with the use cases and requirements for the proposed language extension. In Section 7, we describe the design of the server-side and database-related functionalities to be included into standard XForms. Section 8 presents the implementation details of XFormsDB, a framework supporting the proposed language extension. Finally, we discuss the feasibility of our approach in Section 9 and present our conclusions and suggestions for future work in Section 10.

2. RELATED WORK

An overview of software development methodologies used for Web application development can be found in [5]. Toffetti *et al.* [36] reviewed the current state-of-the-art in RIA development approaches. They indicated that the current framework-based RIA development practices lack support for complete application development (client-side and server-side application logic, client-server communication, and interaction). We addressed this issue by proposing a framework based on a unified development model.

In general, unified Web application development frameworks can be based either on programming, modeling, or markup languages. For instance, *Hop* is a general-purpose Web programming language primarily designed for programming small- and medium-sized interactive Web applications [34, 35]. Hop—with its Scheme-based syntax—exposes a model based on two computation levels: while the application logic is executed on the first level, the second level is responsible for the graphical user interface (GUI). Though Hop separates the application logic from the GUI, it packages them together and supports strong collaboration between them via execution engines. While Hop provides an extensive set of libraries for Web development, its main drawback is not relying on any W3C-standardized language.

Hanus and Kluß presented a declarative multi-paradigm programming language called *Curry* [11] to describe graphical user interfaces for desktop applications as well as Web user interfaces for standard Web browsers. Curry has a Haskell-like syntax and divides the design of a user interface (UI) into three parts: structure (the hierarchical structure of UI elements, such as text inputs or select fields), functionality (the interaction with UI elements), and layout (the visual appearance of UI elements). In comparison with XFormsDB, Curry does not rely on Web standards, and thus is unlikely to be attractive to Web developers. In addition, Curry does not provide any server-side or database-related functionality required in most Web applications.

Kuuskeri and Mikkonen [19] introduced a JavaScript-based middleware platform, extending the JavaScript language with server-side functionality. As in our approach, this proposed Web development model uses one client-side language only. The two approaches differ on the conceptual level: while Kuuskeri and Mikkonen presented a server-side language extension to imperative JavaScript, we expanded the scope of declarative XForms.

Hilda [38] is a Web application development framework based on Unified Modeling Language (UML). The data model is relational, and thus query and update operations use SQL. Hilda's main constructs are AUnits, which correspond to UML classes. However, the presentation layer is based on HTML. The Hilda compiler translates a Hilda program into executable code: a server-side Java Servlet and client-side scripts. Compared to Hilda, XForms simplifies Web application development, since it includes both the data model and the client-side application logic. In addition, the embedding of XForms into a host markup language (e.g., XHTML) is well defined.

Unified Web application development frameworks based on markup languages are usually built either upon XML or HTML (including HTML5). Cardone *et al.* [6] proposed a programming model, which simplifies the design of form-based Web applications by separating client-side XML markup from the server-side programming language considerations. They based their approach on XForms, separating the data representation used on the client (XML) from the programming language structures (Java) native to the server. Unlike Cardone *et al.*, we propose a server-side language extension to XForms that together allow developing a Web application user interface and all of its application logic using XForms only.

Hemel and Visser [13] introduced *mobl*, a high-level, declarative language for programming mobile Web applications. Mobl integrates languages for user interface design, styling, data modeling, querying, and application logic. Before deployment, mobl compiler translates the language into a combination of HTML, CSS, and JavaScript. Compared to XFormsDB, mobl's approach is based more on imperative languages. Furthermore,

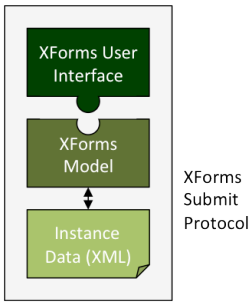


Figure 1. The main components of XForms are based on the MVC architecture.

mobl is not directly based on an existing language, such as XForms. In addition, it does not support server-side data synchronization.

Heinrich and Gaedke [12] proposed the *WebSoDa* framework for binding UI elements and data objects. The WebSoDa framework consists of an HTML5 Microdata binding language as well as both client-side and server-side messaging components. The client-side component is an external JavaScript file. It parses the binding expressions and automatically establishes bidirectional connections to the server-side component using WebSocket as a messaging protocol. The server-side component is implemented using Java Servlet. Compared to XFormsDB, WebSoDa uses more advanced WebSocket for real-time communication, but it lacks of more advanced XFormsDB features, such as error handling, session management, and access control.

3. XFORMS

XForms [4], a W3C recommendation since October 2003, is an XML-based client-side forms technology and the successor to HTML forms. In contrast to conventional HTML forms, an XForms form cleanly separates the presentation, *XForms User Interface*, from the logic, *XForms Model*, and data, *Instance Data*, of a form by internally following the Model-View-Controller (MVC) architecture [17]. Figure 1 illustrates the main components of an XForms form.

Instance Data defines an arbitrary XML document template (the Model part of MVC) for the data to be collected in a form. The initial content and structure of an XML document can be dynamically modified afterwards through user interactions.

XForms Model uses XML to define the non-visual portion—that is, the data and the client-side application logic (the Controller part of MVC)—of a form. The data portion contains one or more *Instance Data* definition(s), whose structures and data types can be defined using XML Schema [8]. The logic portion embodies data submission definitions and Model Item Properties (MIPs) written in XPath [2]. The MIPs define dynamic calculations and constraints on *Instance Data* nodes (e.g., dependencies between various *Instance Data* nodes), which are impossible to define using XML Schema.

XForms User Interface provides a standard control set to declaratively define the visual portion of a form. The form controls (the View part of MVC) are bound to *Instance Data* nodes, allowing the separation between presentation and data.

XForms Submit Protocol defines how XForms sends and receives *Instance Data* as well as the serialization of that data. The data is typically transferred to and from a server, but XForms also allows saving the data to local files, for later reuse.

XForms itself does not define a document format, and therefore must always be embedded into a host language, such as XHTML or SVG. XForms also integrates seamlessly with other declarative W3C standards, including XPath (querying), XML Schema (validation), and CSS (styling). Furthermore, using XForms for authoring dynamic forms does not preclude the use of imperative scripting languages, such as JavaScript, but they can co-exist and interact within the same document.

Currently, only experimental browsers such as X-Smiles[3] [15] support XForms natively. Fortunately, several options are available, ranging from browser plug-ins and client-side XSLT transformations to Ajax-based server-side transformations, that allow XForms to be used in all modern Web browsers.

3.1 Extending XForms

Conventional HTML forms offer limited extensibility options, whereas XForms has been explicitly designed from the start with extensibility in mind. The different options available for extending XForms include *script*, *new data types and libraries*, *XPath extension functions*, *new form controls*, *XForms Actions*, *custom events*, and *new serialization formats* [7]. However, the use of certain XForms extension options does not suit well to be used with XForms implementations relaying on native browser support or browser plug-ins because it requires end users to update the client (XForms processor) running in the browser. XForms also allows foreign attributes in all XForms elements. Foreign elements from any namespace other than XForms, however, can only be used when defined within the `extension` element or in a host language.

3.2 XForms in Web Applications

Figure 2 depicts the conventional three-tier Web application architecture using XForms. The architecture follows the MVC design pattern by distinctively separating the declaratively defined user interface from the application logic and persistent data residing on the server. The user interface consists of documents, which are written in XForms and use XHTML as a host language. The difference between user interface technologies used in today's Web applications is that declarative XHTML is combined with declarative XForms instead of imperative JavaScript. The application logic portion residing on the server, on the other hand, contains custom software components using an application-specific programming language and data model. Typically, the server also hosts a server-side XForms processor in order to ensure cross-browser compatibility. The communication between the client and the server occurs asynchronously over HTTP(S), in which collected form data, *Instance Data*, is serialized in XML and submitted to the server using an HTTP(S) POST request. The benefit of using asynchronous submissions, similarly as in Ajax [10], is that it allows the user interface to remain responsive, while the request is being processed on the server. Finally, the server returns an XML response to the client, and the user interface is dynamically updated according to that response.

[3] X-Smiles, http://www.x-smiles.org/

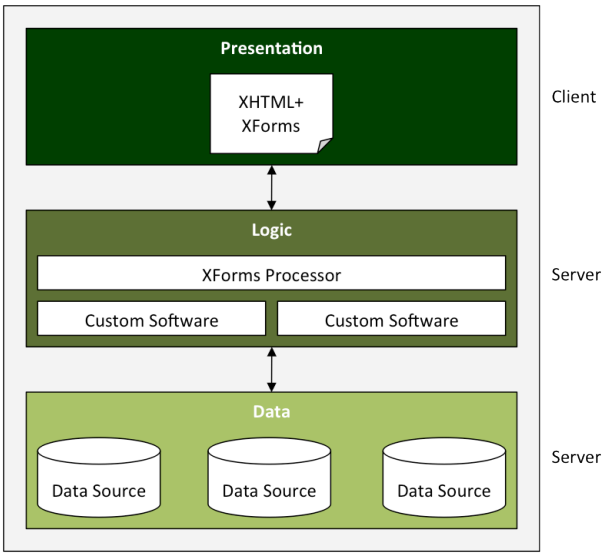


Figure 2. The conventional three-tier Web application architecture using XForms instead of JavaScript for the client-side application logic.

4. XQUERY

XQuery [3] is a declarative query language designed by W3C for extracting and manipulating data from XML documents or any data source that can be viewed as XML, such as relational databases. XQuery has a lot in common with XPath [2], another W3C recommendation for addressing parts of an XML document, as XQuery 1.0 is a superset of XPath 2.0 and they both share the same data model as well as the same set of functions and operators.

XQuery overcomes the limitations of XPath (e.g., lack of grouping, sorting, and cross document joins) by providing an SQL-like feature called a FLWOR expression, in which FLWOR stands for "*for*, *let*, *where*, *order by*, and *return*", the keywords used in the expression. By using FLWOR expressions, it is possible to select and filter XML data based on specific criteria as well as transform and restructure the XML data into another XML vocabulary or structure.

One of the main design goals of XQuery was that it would use and share appropriate W3C standards as much as possible, such as XML (modeling), Namespaces (qualifying), XPath (querying), and XML Schema (validation). In addition, there are several peripheral W3C standards and working drafts that complement XQuery with capabilities, such as updating, full-text searching, and scripting. These complementary specifications, along with existing XQuery extensions, turn XQuery into a general-purpose programming language, powerful enough to replace proprietary server-side programming languages, such as Java. XQuery is widely implemented and supported by native XML databases as well as all major database vendors.

5. EXPANDING THE PRESENTATION TIER

Although this architectural change from JavaScript to XForms simplifies the development process from a Web designer's point of view, there are still significant architectural hurdles to overcome in developing entire Web applications. For example, as Figure 3a shows, in a typical Web application using XForms in conjunction with XHTML, the server-side application logic is implemented using an object-oriented imperative language, such as Java, Ruby, or PHP. The client and the server communicate using declarative formats (e.g., XML or JSON [4]) and asynchronous submissions. In addition, a data-mapping library for translating the data between distinct formats used on the two tiers may be used. Finally, on the undermost tier of the application, i.e., the data tier, either an ORM library or declarative SQL statements manage the data stored in a relational database.

To accomplish all of the aforementioned processes requires tier-specific experts because the programming languages, programming paradigms, and data models differ on each tier. In addition, the manual partitioning of a Web application between the client (presentation tier) and the server (logic and data tiers) complicates the development process. [18, 37]

From a Web designer's point of view, one way of simplifying the Web application architecture is expanding the presentation tier to cover all three tiers. This presentation-centric architectural expansion allows using a single programming language and paradigm—namely, declarative XForms—as well as the XML data model throughout the entire Web application. Figure 3b depicts this presentation-centric architectural approach for extending XForms with common server-side and database-related functionality. The approach follows the MVC design pattern, where XForms can be seen as the View part, its extension as the Controller part, and XPath (part of XForms) as the language for managing the Model part stored in a database.

6. RESEARCH PROBLEM AND SCOPE

This paper explores *how to extend the XForms markup language with common server-side functionality*. The server-side language extension primarily aims to simplify the development and maintenance work of highly interactive data-driven Web applications so that users—mainly Web designers—can implement simple yet useful Web applications quickly and easily using only markup languages. Because most of common server-side functionality relates to data management, researching *how to seamlessly integrate a standardized query language with the XForms markup language* is also important. Covering the functionality of complex Web applications, however, is beyond the scope of this extension, as both the XForms markup language and the server-side extension are targeted at Web designers, who do not require advanced application logic in their Web applications.

6.1 Use Cases

The following subsections describe three possible Web applications, in which the server-side language extension can be utilized.

[4] Possible candidate for XForms 1.2. Specification is available at: http://www.w3.org/MarkUp/Forms/wiki/Json

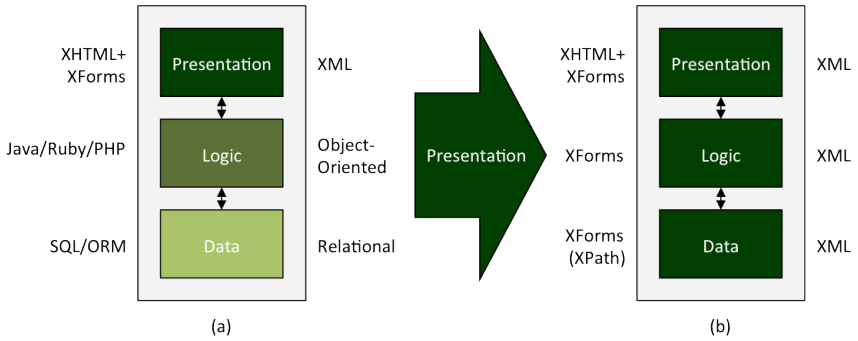


Figure 3. (a) The conventional three-tier Web application architecture using XForms and its (b) presentation-centric architectural expansion.

6.1.1 Address Book

Address Book[5] is a simple application that allows users to store, browse, and manage information about their personal contacts, such as names, addresses, phone numbers, and e-mail addresses. In addition, the list of contacts can be sorted and the language of the user interface can be changed between Finnish, Swedish, and English.

6.1.2 Blog

Blog[6] is an online journal tool for publishing content, such as news, thoughts, comments, and experiences. It allows users to browse through archives, read published posts, and leave comments on the posts. The application offers necessary tools for administrators to write new posts as well as manage published posts and comments.

6.1.3 Project Management

Project Management[7] is a comprehensive software that simplifies project planning, tracking, and management. The software includes sections for managing a user's profile, browsing announcements about news and upcoming events, following projects' deadlines and statuses, sharing documents, and reporting working hours. Functions available on each section are determined by the roles of a currently logged-in user.

6.2 Requirements

Kaufmann and Kossmann [16] listed general requirements for Web applications that cover all three tiers of a Web application, including communication requirements. From this list, only four requirements fall within the scope of the server-side language extension: *persistence and database*, *error handling*, *session management and security*, and *modules to facilitate recurring tasks*. In addition, we included two additional general requirements, which were derived from the use cases: *state maintenance* as well as *authentication, authorization, and access control*. Finally, we defined two specific requirements for the language extension: *similar syntax and processing model* as well as *extensible architecture*.

6.2.1 General Requirements

Persistence and database: A uniform API for connecting to different types of data sources must be provided. In addition, a standardized declarative query language, which is applicable across all data sources viewable as XML, must be supported.

Error handling: A method for notifying the client about errors occurred while processing a requested server-side command must be provided.

Session management and security: Managing sessions between the client and the server must be supported regardless of the browser used or its settings. In addition, documents sent to the client must neither expose nor allow the unauthorized alteration of sensitive information.

Modules to facilitate recurring tasks: A method to facilitate modularity and the reuse of ready-made components (e.g., user interface parts and queries) in Web applications must be supported.

State maintenance: A method to maintain the state in Web applications—especially a mechanism for passing state information (e.g., *Instance Data*) between documents—must be supported.

Authentication, authorization, and access control: A simple way to authenticate users and to handle common access control tasks must be provided.

6.2.2 Language Extension Requirements

Similar syntax and processing model: The syntax and processing model of the server-side language extension must be similar to XForms.

Extensible architecture: The architecture for the server-side language extension must provide a method to define new features—that is, server-side commands—while retaining the same processing model.

[5] Address Book, http://testbed.tml.hut.fi/pim/

[6] Blog, http://testbed.tml.hut.fi/blog/

[7] Project Management, http://flexi.tml.hut.fi/fs/

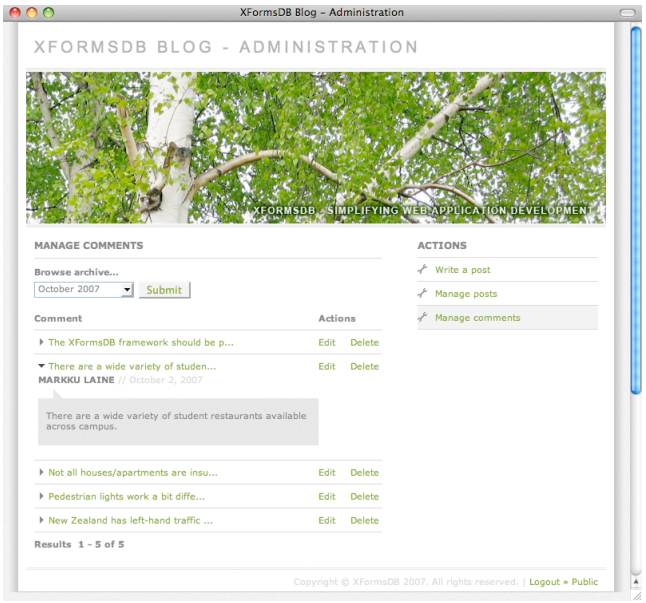


Figure 4. A screenshot image showing the Blog administration user interface.

7. DESIGN OF THE LANGUAGE EXTENSION

This section provides a high-level description of the proposed server-side language extension, which was designed to meet the research objectives and requirements presented in Section 6. The language extension specifies common server-side and database-related functionalities, which are needed to turn XForms into a comprehensive Web programming language. The following defines the functionalities provided by the language extension: *definition of server-side requests*, *submission of server-side requests*, *notification about server-side errors*, *permission management*, and *reuse of code fragments*.

We chose the Blog Web application (cf. Figure 4) from the use cases to demonstrate the applicability of the proposed server-side language extension, which is used as an example throughout the rest of this paper. Listing 1 shows an excerpt of the relevant portions of the application source code[8]. The namespace URI for the language extension used in the code example is `http://www.tml.hut.fi/2007/xformsdb` and is bound to the prefix *xformsdb*. The complete description of the proposed language extension, along with syntax definitions and usage examples, is available in [20].

7.1 Definition of Server-Side Requests

Server-side requests are commands submitted to the server, where they are securely executed. They are defined within a new element, `xformsdb:instance`, that acts as a wrapper for all server-side requests. The benefit of using a wrapper around server-side requests is that it enables adding new features to the language without requiring any changes to the request processing model. Currently, the language extension includes definitions for the following server-side commands: *maintaining state information*, *logging users in* and *out*, *retrieving information about a currently logged-in user*, *executing queries against data sources*, *managing files*, and *checking the browser support for cookies*.

The demonstrated application utilizes three of the aforementioned commands: *logging users in* and *out* for authenticating blog administrators and *executing queries against data sources*. Lines 27-34 show an example definition of a *query* command for retrieving the comments of a specific blog post, identified by an external variable *$postid*. In this particular example, the parameterized query expression is written in XPath and defined in an external resource (cf. line 30). The query expression is executed against a single XML document (`blog.xml`) stored in a database, when a corresponding submission is dispatched.

7.2 Submission of Server-Side Requests

The `xformsdb:submission` element is a new element that can submit server-side requests that have been defined within the same document. As with the standard XForms submissions, server-side requests can also be submitted multiple times and at any point in a form's lifetime.

The demonstrated application has multiple server-side request submission elements for submitting various commands to be executed on the server. Lines 35-49 show how the *query* command defined in the previous subsection can be submitted to the server, where it is securely executed against the `blog.xml` document stored in the database.

For triggering the submission, the standard XForms `send` action is used. After a successful submission, the query result extracted from the database is stored in an XForms `instance` element (cf. lines 23-26), whose original content is replaced with the extracted data. Finally, lines 67-70 iterate over the data within the aforementioned XForms `instance` element and display it in the main content area of the Blog administration user interface.

7.3 Notification about Server-Side Errors

XForms includes a set of different events (e.g., `xforms-ready` and `xforms-submit-done`), which can be caught by standard XForms event handlers (XForms Actions) using XML Events. XForms also provides a possibility to create custom events. We have extended this set of predefined events to include a new notification-type event, `xformsdb-request-error`, that is dispatched to indicate a failure in a server-side request submission and/or execution process. For example, the event is dispatched if an error occurs in establishing a connection to a data source or in executing a query expression. The event's context information, i.e., the error code and description, can be accessed with the XForms `event` function.

In the demonstrated application, the `xformsdb-request-error` event is used within all `xformsdb:submission` elements to catch server-side errors, as shown in lines 45-48.

[8] For the demonstration and readability reasons, the application source code (available at http://tinyurl.com/xformsdb-blog-src) may differ from the code shown in Listing 1.

```
<?xml version="1.0" encoding="UTF-8"?>
<html xml:lang="en" lang="en"
  xmlns="http://www.w3.org/1999/xhtml"
  xmlns:ev="http://www.w3.org/2001/xml-events"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xforms="http://www.w3.org/2002/xforms"
  xmlns:xformsdb="http://www.tml.tkk.fi/2007/xformsdb"
  xmlns:xxforms="http://orbeon.org/oxf/xml/xforms">
  <head>
    <title>XFormsDB Blog - Administration</title>
    <!-- Include common metadata information -->
    <xformsdb:include resource="../xinc/meta.xinc" />
    <!-- XFormsDB security view for non-logged in users -->
    <xformsdb:secview>
      <xforms:model>
        <xforms:load resource="../login.xformsdb" ev:event="xforms-ready" />
      </xforms:model>
    </xformsdb:secview>
    <!-- XFormsDB security view for logged in users having one of the roles: admin -->
    <xformsdb:secview roles="admin">
      <xforms:model>
        <!-- Placeholder for comments -->
        <xforms:instance id="comments">
          <dummy xmlns="" />
        </xforms:instance>
        <!-- Definition of the select and update comments query command -->
        <xformsdb:instance id="select-and-update-comments">
          <xformsdb:query datasrc="exist-db" doc="blog.xml">
            <xformsdb:expression resource="../xpath/select_and_update_comments.xpath" />
            <xformsdb:xmlns prefix="xformsdb" uri="http://www.tml.tkk.fi/2007/xformsdb" />
            <xformsdb:var name="postid" />
          </xformsdb:query>
        </xformsdb:instance>
        <!-- Definition of the select comments query command submission -->
        <xformsdb:submission id="sub-select-comments" replace="instance" instance="comments"
          requestinstance="select-and-update-comments" expressiontype="select">
          <xforms:action ev:event="xforms-submit">
            <xforms:toggle case="on-progress-animation" />
          </xforms:action>
          <xforms:action ev:event="xforms-submit-done">
            <xforms:toggle case="off-progress-animation" />
            <xforms:toggle case="manage-comments-ui" />
          </xforms:action>
          <xforms:action ev:event="xformsdb-request-error">
            <xforms:toggle case="off-progress-animation" />
            <xforms:toggle case="sub-select-comments-error-message" />
          </xforms:action>
        </xformsdb:submission>
        ...
      </xforms:model>
    </xformsdb:secview>
  </head>
  <body>
    <!-- XFormsDB security view for non-logged in users -->
    <xformsdb:secview>
      <xforms:trigger appearance="minimal">
        <xforms:label>Redirecting...</xforms:label>
        <xforms:action ev:event="DOMActivate">
          <xforms:load resource="../login.xformsdb" />
        </xforms:action>
      </xforms:trigger>
    </xformsdb:secview>
    <!-- XFormsDB security view for logged in users having one of the roles: admin -->
    <xformsdb:secview roles="admin">
      ...
      <!-- Iterate over comments -->
      <xforms:repeat nodeset="instance( 'comments' )/comment" id="repeat-comments-comment">
        ...
      </xforms:repeat>
      ...
    </xformsdb:secview>
  </body>
</html>
```

Listing 1. An excerpt of the application source code of the Blog administration user interface.

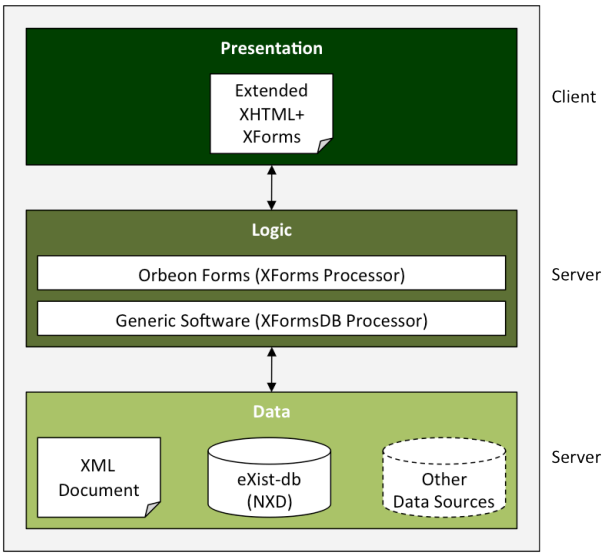


Figure 5. The high-level architecture of the XFormsDB framework.

7.4 Permission Management

Standard XForms does not provide a secure mechanism for controlling user access to certain portions of a document. We have extended XForms to include a role-based authorization system. The system contains (1) a new element, `xformsdb:secview`, to control user access within a document; (2) server-side requests for *logging users in* and *out*, and *retrieving information about a currently logged-in user*; and (3) a realm XML document representing a "database" of usernames, passwords, and roles assigned to those users.

The demonstrated application uses the authorization system to control user access to the Blog administration user interface. Only users with the *admin* role can access the webpage, whereas others are redirected to the login webpage. Lines 14-19, 20-52, 55-63, and 64-72 demonstrate the use of the `xformsdb:secview` element.

7.5 Reuse of Code Fragments

The `xformsdb:include` element is a new element that provides a recursive inclusion mechanism to facilitate modularity. This element allows the construction of large XML documents from several well-formed XML documents. The idea behind the `xformsdb:include` element differs from XInclude [25] only in that its processing model lines up with the other new elements and is simpler than XInclude.

In the demonstrated application, the `xformsdb:include` element includes common metadata information on all webpages (cf. lines 12-13).

8. IMPLEMENTATION

The Blog Web application presented above was implemented using a framework called *XFormsDB* [20]. The XFormsDB framework is an open source project and is available at `http://code.google.com/p/xformsdb/`. The framework is implemented in pure Java, and includes an XFormsDB processor supporting the proposed server-side language extension. The architecture of the XFormsDB framework and the XFormsDB processor is presented below as a reference implementation of the proposed language extension.

8.1 The XFormsDB Framework

The XFormsDB framework is a generic platform for developing and hosting Web applications based on the XForms markup language and its server-side extension, as proposed in this paper. The framework uses a set of third-party software and libraries, including the *Apache Tomcat*[9] HTTP Web server, the *eXist-db*[10] native XML database (NXD) [28], and the *Orbeon Forms*[11] Ajax-based server-side XForms processor.

Figure 5 depicts the high-level architecture of the XFormsDB framework. Here, it differs from the conventional three-tier Web application architecture using XForms (cf. Figure 2) in that a generic software component (an XFormsDB processor) replaces the functionality provided by custom server-side software components. Because of this architectural change, all application development is now moved to the client side and is performed in extended XHTML+XForms documents. The server also hosts an Ajax-based server-side XForms processor called Orbeon Forms, which in the end—if necessary—transforms these documents into cross-browser (X)HTML+CSS+JavaScript or plain (X)HTML+CSS, depending on the configuration. The communication between the client and the server happens asynchronously over HTTP(S). Currently, the framework supports only XML-based data sources (XML documents and eXist-db) but by using a middleware, e.g., DataDirect XQuery[12], support for other data sources (e.g., relational databases) can be easily added. In the demonstrated application, all data was stored in the eXist-db native XML database.

8.2 The XFormsDB Processor

The XFormsDB processor is a generic software component supporting the proposed server-side language extension. The processor's responsibilities include *handling requests* and *writing responses*, *transforming extended XHTML+XForms documents*, *managing sessions*, *performing synchronized updates*, and *providing integration services to heterogeneous data sources*. Separate components carry out each of these tasks, as depicted in Figure 6.

When a client makes an HTTP(S) request to the server, the request first reaches the XFormsDB processor and is handled by its front controller, *XFormsDB Servlet*. The front controller extracts relevant request information and forwards the request to an appropriate request handler. In case an extended XHTML+XForms document is requested, *XFormsDB Transformer* processes the document according to the following steps:

[9] Apache Tomcat, http://tomcat.apache.org/

[10] eXist-db, http://exist-db.org/

[11] Orbeon Forms, http://www.orbeon.com/

[12] DataDirect XQuery, http://www.xquery.com/xquery/

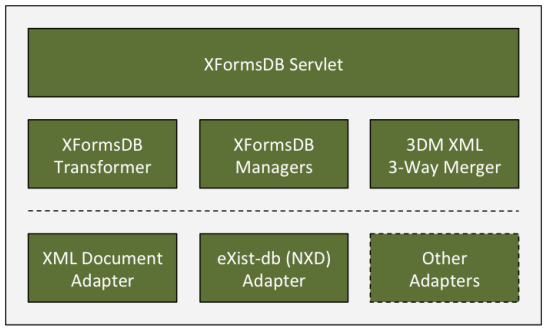


Figure 6. The components of the XFormsDB processor.

1. Parse the document and identify server-side extension elements.
2. Incorporate all external documents into the main document (cf. `xformsdb:include`).
3. Filter out those parts to which the user does not have access rights (cf. `xformsdb:secview`).
4. Identify and collect information from other relevant elements (e.g., `xformsdb:instance`, `xformsdb:query`, and `xformsdb:submission`).
5. Store the collected information found in step 4 in the session (*XFormsDB Managers*).
6. Transform the document (including the server-side extension elements) into XHTML+XForms 1.1 compliant markup, in which the definitions of server-side commands containing sensitive information have been substituted with opaque reference IDs for security reasons. During the transformation process, certain utility instances (e.g., an *Instance Data* containing HTTP request headers) are automatically added to the document.
7. Return the transformed document.

Before returning the transformed document to the client, the document goes through another transformation process (cf. Orbeon Forms) that transforms it into a format viewable by the requesting client.

Asynchronous form submissions over HTTP(S) also go through the front controller (*XFormsDB Servlet*), which extracts relevant request information and forwards the request to an appropriate request handler based on the submitted command. In the case of a *query* command submission, the original query expression is fetched from the session (*XFormsDB Managers*) using the opaque reference ID submitted along with the *query* command, and then executed against the underlying data source (*XML Document* and *eXist-db Adapters*). Finally, a response XML is composed and returned to the client.

8.2.1 Data Synchronization

The XFormsDB processor includes built-in support for performing synchronized updates (*3DM XML 3-Way Merger*). To accomplish data synchronization, the XFormsDB processor uses 3DM[13] [23], a middleware for performing three-way merging of XML documents, which is able to detect and handle *update*, *insert*, and *delete* operations as well as *moves* and *copies* of entire subtrees. Furthermore, the aforementioned operations can be performed without the use of unique element identifiers, i.e., original XML documents can be used as such without equipping them with excess attributes.

We illustrate how the 3DM merging process works in the example shown in Figure 7. In the example, (a) is referred to as the original version, (b) as the altered version, (c) as the current version stored in the data source, and (d) as the merged version. Green color indicates that the node has been either updated (marked with an asterisk), inserted, or moved, whereas white color indicates that the node has remained unaltered.

In XFormsDB, the updating process *with* data synchronization includes the following steps. In the first step, an XML fragment is retrieved from a data source using an XPath expression that points to the root element of the XML fragment to be updated. Then, the retrieved XML fragment can be altered on the client, after which the altered XML fragment is submitted back to be stored in the data source using the same XPath expression as before. Next, the data synchronization process is performed and upon a successful synchronization, the result XML fragment is stored in the data source. Finally, the stored XML fragment, which may contain changes made concurrently by other clients, is returned to the client. In case the data synchronization process fails (e.g., a merge conflict), an appropriate error message is reported back to the client, which handles the error on a case-by-case basis.

8.3 Extensibility and Limitations

The XFormsDB framework supports extensibility at different levels of the architecture. The most elegant way of extending the architecture is by defining new server-side requests to the language extension, as stated in Subsection 7.1. Listing 2 shows a simple example of how to define a new server-side request (cf. line 2) for retrieving HTTP request headers from a hosting server. In the case of the Blog Web application, the retrieved information could be used, for instance, to detect mobile clients and redirect them automatically to the mobile-optimized version of a particular webpage.

```
01. <xformsdb:instance id="http-request-headers">
02.   <xformsdb:httprequestheaders />
03. </xformsdb:instance>
```

Listing 2. An example of a new server-side request definition.

The main disadvantage of this approach is that every syntax addition made to the language extension also needs to be implemented in the XFormsDB processor. In its current state, extended XHTML+XForms documents provide the basic means for users to implement simple yet useful Web applications using only markup languages. Due to the limited expressive power of the languages, however, they are alone insufficient to meet the requirements of more complex Web applications. In the following subsections, we give examples of how the expressive power of the

[13] 3DM, http://developer.berlios.de/projects/tdm/

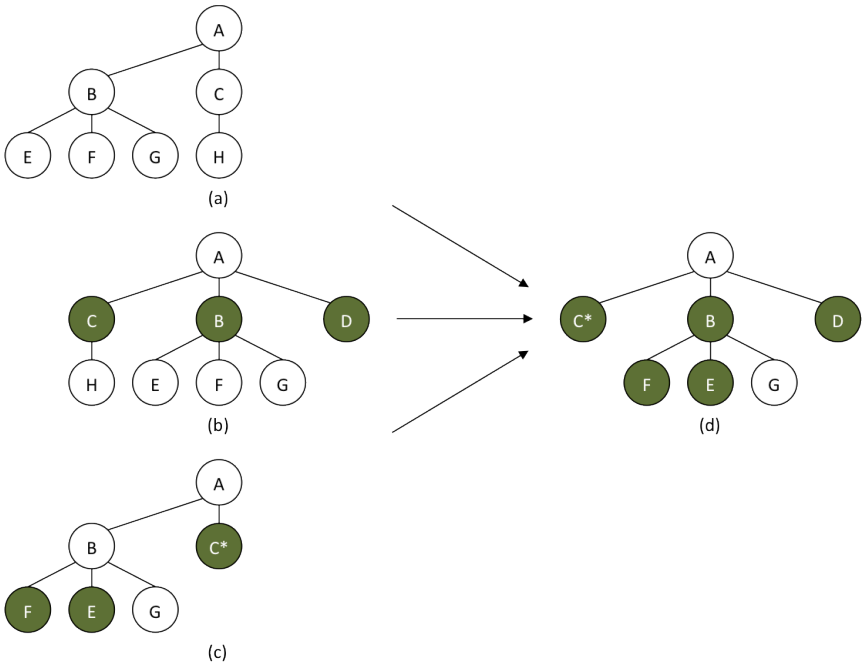


Figure 7. The data synchronization process: a three-way merge for XML documents.

XFormsDB framework can be improved by using different technologies specific to each tier.

8.3.1 Presentation Tier

The XFormsDB framework relies mainly on XForms on the presentation tier. XForms uses declarative markup to define both the form controls and related client-side application logic of a Web application. It also offers a wide variety of options for extensibility, as described in Subsection 3.1.

The primary reasons for using extensions on the presentation tier include adding animations, interactivity, and client-side application logic to webpages that goes beyond the capabilities of XForms. In the demonstrated application, extended XHTML+XForms documents are supplemented with imperative JavaScript embeddings for the purpose of measuring the response time of each webpage. In addition, one of Orbeon Forms' XForms extension attributes is used to format dates into a human-readable form (cf. Listing 3).

```
01. <xforms:output ref="creationtime"
02.   xxforms:format="format-date( xs:date( . ),
03.   '[MNn] [D], [Y]', 'en', (), () )" />
```

Listing 3. An example of Orbeon Forms' XForms extension attribute used in the `xforms:output` element.

8.3.2 Logic Tier

Our proposed server-side extension to XForms is responsible for covering common server-side and database-related functionality required by most small-sized Web applications. To meet more advanced server-side application logic requirements, XQuery (cf. Section 4), and especially eXist-db's XQuery extension functions[14] can be utilized. eXist-db's functions are divided into pluggable modules and, as a whole, provide an extensive set of functionality ranging from small utility functions (e.g., for performing date and time operations as well as transforming XML into JSON) to complete libraries, such as the HTTP Client module.

In the Blog Web application, we did not resort to any eXist-db's XQuery extension modules. Nonetheless, we demonstrate an example of how an extension module could be used in XQuery code (cf. Listing 4).

```
01. xquery version "1.0" encoding "UTF-8";
02. declare namespace httpclient =
03.   "http://exist-db.org/xquery/httpclient";
04. ...
05. httpclient:get( $url, $persist, $requestHeaders )
```

Listing 4. An example of eXist-db's XQuery extension module used in XQuery code.

8.3.3 Data Tier

As described in Subsection 8.2.1, the XFormsDB framework provides a simple and elegant means for performing queries and synchronized updates using XPath only. Though this approach has its advantages, it also inherits few problems to be addressed. For instance, performing a simple *insert* or *delete* operation requires redundant transfer of large XML fragments between the client and

[14] eXist-db's XQuery extension modules, http://exist-db.org/exist/extensions.xml

the server. In addition, an XML fragment that needs to be updated might expose sensitive information to the client.

To overcome the limitations of the aforementioned method, and XPath in general, the XFormsDB framework provides an option to use a more expressive query language, XQuery. The demonstrated application relies heavily on XQuery and its standard functions as well as the functions defined in the *FunctX XQuery Function library* [15]. Listing 5 shows an XQuery expression which retrieves a single blog post identified by the external variable *$id* and returns a custom XML document as a response.

```
01. xquery version "1.0" encoding "UTF-8";
02. declare namespace xformsdb =
03.   "http://www.tml.tkk.fi/2007/xformsdb";
04. declare variable $id as xs:string external;
05.
06. for $post in /root/blog/posts/post
07.   where $post/@id = $id
08.   return
09.     <post id="{ $post/@id }">
10.       <headline>{ $post/headline/text() }</headline>
11.       <creationtime>
12.         { $post/creationtime/text() }
13.       </creationtime>
14.       <content>{ $post/content/text() }</content>
15.       <author>{ $post/author/text() }</author>
16.       <comments>
17.         { count( $post/comments/comment ) }
18.       </comments>
19.     </post>
```

Listing 5. An XQuery expression for retrieving a specific blog post identified by its *$id*.

Besides the extension methods discussed above, the XFormsDB framework allows the use of the same XML Schema document both on the client and the server to validate the structures and data types of transmitted XML documents. Furthermore, with the extension methods, the framework becomes fully compatible with the XRX (XForms/REST/XQuery) architecture [26], making it a viable option for developing complex Web applications.

An obvious drawback of each extension method is, however, that they all require a user to learn a new technology. The framework also shares some of the problems that are common to many Ajax-based Web applications, such as problems related to the use of the browser's back button and bookmarking. In addition, the XFormsDB framework assumes that the user possesses a basic knowledge of XForms and our proposed server-side language extension, which may be a barrier for Web content authors. Fortunately, these limitations can be addressed by providing a Web-based tool that allows authors to visually develop XFormsDB-based Web applications. The prototype of such a tool called *XFormsDB IDE (XIDE)*[16] has already been implemented and the results have been published in a separate paper [24].

[15] FunctX XQuery Function Library, http://www.xqueryfunctions.com/

[16] XFormsDB IDE (XIDE), http://code.google.com/p/xformsdb-ide/

9. DISCUSSION

Typically, the amount of code in Web applications is distributed approximately equally between the client and the server [16]. In the Blog Web application that uses the XFormsDB framework, this ratio was 90% and 10% respectively [20], meaning that our proposed server-side language extension (including XPath and XQuery code) significantly reduced the amount of code required to develop the server-side application logic and data management functionalities. In terms of lines of code (LoC), this means that the amount of code required for implementing the Blog Web application was decreased by 45%—that is, approximately 2400 LoC[17]. Detailed metrics for the Address Book and Blog Web applications (cf. Sections 6.1.1 and 6.1.2, respectively) are available in [20].

The advantages of using the XRX architecture compared to the conventional three-tier architecture are further discussed in [31]. In their paper, Nemeş *et al.* show that applications developed according to the XRX architecture are more efficient and elegant. They continue by stating that the XRX architecture increases productivity and reduces implementation costs.

According to Cardone *et al.* [6], there are three main reasons for the inefficiency of the conventional three-tier architecture when it comes to developing complex Web applications. First, dynamic webpages are often generated on the fly, making application source code harder to understand and debugging more difficult. Second, dynamic webpages often contain a mixture of markup languages, client-side scripting code, and server-side function calls, making application source code nearly unreadable and difficult to maintain. Third, the high number of tools, technologies, and techniques used in developing Web applications makes those applications complicated to design and fragile to deploy and run.

Using only one language on all three tiers reduces the number of technologies involved and can unify the Web development process [21]. Determining the most suitable language for building Web applications thus becomes a question. According to Schmitz [33], declarative languages (e.g., XHTML) have several advantages over imperative languages (e.g., Java). Particularly, one compelling advantage is that most Web content authors, not being programmers, prefer declarative languages. Moreover, content authors working on the presentation tier are, as a rule, familiar with declarative (X)HTML and CSS, but not the server-side aspects of a Web application. They can thus benefit from a client-side language that has been extended with server-side functionality.

To justify the choice of our client-side programming language, we followed a recent survey [32], in which five XML-based client-side languages, including HTML5 [14] and XForms, were evaluated. According to the study, XForms is best suited for data-intensive applications and applications with accessibility requirements. XForms also provides a rich declarative use of client-side data and can easily define interdependencies between the data and user interface.

[17] Note that these results are merely suggestive, as the application has not been developed using the conventional three-tier Web application architecture.

10. CONCLUSIONS AND FUTURE WORK

In this paper, we addressed the challenge of simplifying the development and maintenance work of highly interactive data-driven Web applications. We proposed a presentation-centric architectural approach that allows users to implement simple yet useful Web applications using only markup languages. The approach is based on a W3C-standardized client-side technology, the XForms markup language, and its server-side extension proposed in this paper. In addition, we presented the XFormsDB framework, a comprehensive implementation of the proposed approach and the language extension. The framework has four major advantages. First, the entire application can be developed using a single markup language and within a single document (i.e., an extended XHTML 2.0 document). Second, the same data model (i.e., XML) can be used across all tiers of a Web application. Third, the approach is based on declarative programming, and thus allows users with limited programming skills (particularly Web user interface designers who are well familiar with declarative HTML and CSS) to create entire Web applications. Finally, the framework offers a variety of options for extensibility, mainly by leveraging declarative W3C standards. This framework, together with a number of examples, is available under the MIT license at `http://code.google.com/p/xformsdb/`.

In our future work, we will mainly focus on the next two functionality aspects: (1) real-time communication and (2) client-side storage. By adding real-time communication capabilities (namely, XMPP over WebSocket [30] together with a declarative API definition) to the XFormsDB framework, opens up new possibilities for developing more dynamic, event-driven Web applications. To adapt the XFormsDB framework to the requirements of mobile Web application development, we plan to extend our framework with the support for client-side databases (e.g., IndexedDB [27]). Together these technologies can yield significant improvements in performance and user experience for highly interactive data-driven Web applications.

11. ACKNOWLEDGMENTS

This research work was conducted as part of TIVIT's Flexible Services program and its Ecosystem Design and Evolution (EDEN) project. The funding for this project was granted by Tekes and Nokia Research Center.

12. REFERENCES

[1] Alonso, G., Casati, F., Kuno, H., and Machiraju, V. *Web Services: Concepts, Architectures and Applications (1st ed.)*. Springer, 2004. ISBN: 978-3-540-44008-6.

[2] Berglund, A., Boag, S., Chamberlin, D., Fernández, M.F., Kay, M., Robie, J., and Siméon, J. (eds.). *XML Path Language (XPath) 2.0 (Second Edition)*. W3C Recommendation, December 2010. http://www.w3.org/TR/xpath20/.

[3] Boag, S., Chamberlin, D., Fernández, M.F., Florescu, D., Robie, J., and Siméon, J. (eds.). *XQuery 1.0: An XML Query Language (Second Edition)*. W3C Recommendation, December 2010. http://www.w3.org/TR/xquery/.

[4] Boyer, J.M. (eds.). *XForms 1.1*. W3C Recommendation, October 2009. http://www.w3.org/TR/xforms/.

[5] Brandon, D.M. *Software Engineering for Modern Web Applications: Methodologies and Technologies (1st ed.)*. IGI Global, 2008. ISBN: 978-1-599-04492-7.

[6] Cardone, R., Soroker, D., and Tiwari, A. Using XForms to Simplify Web Programming. In *Proceedings of the 14th International Conference on World Wide Web (WWW '05)*, pages 215-224. ACM, 2005. DOI: 10.1145/1060745.1060780.

[7] Dubinko, M. *XForms Essentials (1st ed.)*. O'Reilly Media, 2003. ISBN: 978-0-596-00369-2.

[8] Fallside, D.C. and Walmsley, P. (eds.). *XML Schema Part 0: Primer Second Edition*. W3C Recommendation, October 2004. http://www.w3.org/TR/xmlschema-0/.

[9] Fraternali, P., Rossi, G., and Sánchez-Figueroa, F. Rich Internet Applications. *IEEE Internet Computing*, Vol. 14, No. 3, pages 9-12. IEEE, 2010. DOI: 10.1109/MIC.2010.76.

[10] Garrett, J.J. *Ajax: A New Approach to Web Applications*. February 2005. http://www.adaptivepath.com/ideas/e000385.

[11] Hanus, M. and Kluß, C. Declarative Programming of User Interfaces. In *Practical Aspects of Declarative Languages (PADL '09)*, LNCS 5418, pages 16-30. Springer, 2009. DOI: 10.1007/978-3-540-92995-6_2.

[12] Heinrich, M. and Gaedke, M. Data Binding for Standard-based Web Applications. In *Proceedings of the 27th Annual ACM Symposium on Applied Computing (SAC '12)*, pages 652-657. ACM, 2012. DOI: 10.1145/2245276.2245402.

[13] Hemel, Z. and Visser, E. Declaratively Programming the Mobile Web with Mobl. In *Proceedings of the 2011 ACM International Conference on Object Oriented Programming Systems Languages and Applications (OOPSLA '11)*, pages 695-712. ACM, 2011. DOI: 10.1145/2076021.2048121.

[14] Hickson, I. (eds.). *HTML5: A vocabulary and associated APIs for HTML and XHTML*. W3C Working Draft, March 2012. http://www.w3.org/TR/html5/.

[15] Honkala, M. and Vuorimaa, P. XForms in X-Smiles. *World Wide Web*, Vol. 4, No. 3, pages 151-166. Springer (formerly Kluwer), 2001. DOI: 10.1023/A:1013853416747.

[16] Kaufmann, M. and Kossmann, D. *Developing an Enterprise Web Application in XQuery*. Technical Report. ETH Zürich, 2008. http://download.28msec.com/sausalito/technical_reading/enterprise_webapps.pdf.

[17] Krasner, G.E. and Pope, S.T. A Cookbook for Using the Model-View-Controller User Interface Paradigm in Smalltalk-80. *Journal Of Object-Oriented Programming*, Vol. 1, No. 3, pages 26-49. SIGS, 1988.

[18] Kuuskeri, J. and Mikkonen, T. Partitioning Web Applications Between the Server and the Client. *Journal of Web Engineering*, Vol. 9, No. 3, pages 207-226. Rinton Press, 2010.

[19] Kuuskeri, J. and Mikkonen, T. REST Inspired Code Partitioning with a JavaScript Middleware. In *Current Trends in Web Engineering (ICWE 2010 Workshops)*, LNCS 6385, pages 244-255. Springer, 2010. DOI: 10.1007/978-3-642-16985-4_22.

[20] Laine, M. *XFormsDB—An XForms-Based Framework for Simplifying Web Application Development*. M.Sc. Thesis. Aalto University, January 2010. http://lib.tkk.fi/Dipl/2010/urn100141.pdf.

[21] Laine, M., Shestakov, D., Litvinova, E., and Vuorimaa, P. Toward Unified Web Application Development. *IT Professional*, Vol. 13, No. 5, pages 30-36. IEEE, 2011. DOI: 10.1109/MITP.2011.55.

[22] Laine, M., Shestakov, D., and Vuorimaa, P. XFormsDB: A Declarative Web Application Framework. In *Web Engineering (ICWE 2012)*, LNCS 7387, pages 477-480. Springer, 2012. DOI: 10.1007/978-3-642-31753-8_48.

[23] Lindholm, T. A Three-Way Merge for XML Documents. In *Proceedings of the 2004 ACM Symposium on Document Engineering (DocEng '04)*, pages 1-10. ACM, 2004. DOI: 10.1145/1030397.1030399.

[24] Litvinova, E., Laine, M., and Vuorimaa, P. XIDE: Expanding End-User Web Development. In *Proceedings of the Eighth International Conference on Web Information Systems and Technologies (WEBIST '12)*, pages 123-128. SciTePress, 2012.

[25] Marsh, J., Orchard, D., and Veillard, D. (eds.). *XML Inclusions (XInclude) Version 1.0 (Second Edition)*. W3C Recommendation, November 2006. http://www.w3.org/TR/xinclude/.

[26] McCreary, D. *Introducing the XRX Architecture: XForms/REST/XQuery*. December 2007. http://datadictionary.blogspot.com/2007/12/introducing-xrx-architecture.html.

[27] Mehta, N., Sicking, J., Graff, E., Popescu, A., and Orlow, J. (eds.). *Indexed Database API*. W3C Working Draft, May 2012. http://www.w3.org/TR/IndexedDB/.

[28] Meier, W. eXist: An Open Source Native XML Database. In *Web, Web-Services, and Database Systems (Web Databases and Web Services 2002)*, LNCS 2593, pages 169-183. Springer, 2003. DOI: 10.1007/3-540-36560-5_13.

[29] Mikkonen, T. and Taivalsaari, A. *Web Applications - Spaghetti Code for the 21st Century*. Technical Report, SMLI TR-2007-166. Oracle (formerly Sun Microsystems), June 2007. https://labs.oracle.com/techrep/2007/smli_tr-2007-166.pdf.

[30] Moffit, J. and Cestari, E. (eds.). *An XMPP Sub-protocol for WebSocket*. Internet Draft (Standards Track), June 2012. https://tools.ietf.org/html/draft-moffitt-xmpp-over-websocket-01.

[31] Nemeş, C., Podean, M, and Rusu, L. XRX: The Implementation Process under XRX Architecture. In *Proceedings of the Eighth International Conference on Web Information Systems and Technologies (WEBIST '12)*, pages 103-109. SciTePress, 2012.

[32] Pohja, M. Comparison of Common XML-Based Web User Interface Languages. *Journal of Web Engineering*, Vol. 9, No. 2, pages 95-115. Rinton Press, 2010.

[33] Schmitz, P. *The SMIL 2.0 Timing and Synchronization Model: Using Time in Documents*. Technical Report, MSR-TR-2001-01. Microsoft Research, January 2001. https://research.microsoft.com/pubs/69839/tr-2001-01.doc.

[34] Serrano, M. Programming Web Multimedia Applications with Hop. In *Proceedings of the 15th International Conference on Multimedia (MULTIMEDIA '07)*, pages 1001-1004. ACM, 2007. DOI: 10.1145/1291233.1291450.

[35] Serrano, M., Gallesio, E., and Loitsch, F. Hop, a Language for Programming the Web 2.0. In *Proceedings of the First Dynamic Languages Symposium (DLS '06), Companion to OOPSLA 2006*, pages 975-985. ACM, 2006. DOI: 10.1145/1176617.1176756.

[36] Toffetti, G., Comai, S., Preciado, J.C., and Linaje, M. State-of-the-Art and Trends in the Systematic Development of Rich Internet Applications. *Journal of Web Engineering*, Vol. 10, No. 1, pages 70-86. Rinton Press, 2011.

[37] Yang, F., Gupta, N., Gerner, N., Qi, X., Demers, A., Gehrke, J., and Shanmugasundaram, J. A Unified Platform for Data Driven Web Applications with Automatic Client-Server Partitioning. In *Proceedings of the 16th International Conference on World Wide Web (WWW '06)*, pages 341-350. ACM, 2007. DOI: 10.1145/1242572.1242619.

[38] Yang, F., Shanmugasundaram, J., Riedewald, M., Gehrke, J., and Demers, A. Hilda: A High-Level Language for Data-Driven Web Applications. In *Proceedings of the 22nd International Conference on Data Engineering (ICDE '06)*, pages 32-43. IEEE, 2006. DOI: 10.1109/ICDE.2006.75.

ABOUT THE AUTHORS:

Markku Laine is a doctoral student in the Department of Media Technology at the Aalto University, Finland. He received his M.Sc. degree in Communications Engineering from the Aalto University in 2010. His current research interests include declarative Web application development, Web performance optimization, and real-time communication protocols. Contact him at markku.laine@aalto.fi.

Denis Shestakov is a postdoctoral researcher in the Department of Media Technology at the Aalto University, Finland. He earned his Ph.D. (2008) degree from the University of Turku, Finland. His dissertation addressed the limitations of Web crawlers, specifically the poor coverage of information available in online databases (a.k.a. the Deep Web). His current research interests span the areas of distributed data management and big data processing, with a particular focus on scalable Web agents and services. Contact him at denis.shestakov@aalto.fi.

Petri Vuorimaa is a full professor in the Department of Media Technology at the Aalto University, Finland. He obtained both his M.Sc. (1990) and D.Sc. (1995) degrees from the Tampere University of Technology, Finland. His current research interests include Web-based services, smart spaces, and mobile media applications. Contact him at petri.vuorimaa@aalto.fi.

Analysis of a Triploid Genetic Algorithm over Deceptive and Epistatic Landscapes

Menglin Li
College of Engineering and Informatics
National University of Ireland Galway
m.li1@nuigalway.ie

Seamus Hill
College of Engineering and Informatics
National University of Ireland Galway
seamus.hill@nuigalway.ie

Colm O'Riordan
College of Engineering and Informatics
National University of Ireland Galway
colm.oriordan.nuigalway.ie

ABSTRACT

This paper[1] examines the performance of a canonical genetic algorithm (CGA) against that of the triploid genetic algorithm (TGA) introduced in [14], over a number of well known deceptive landscapes and a series of NK landscapes in order to increase our understanding of the the TGA's ability to control convergence. The TGA incorporates a mechanism to control the convergence direction instead of simply increasing the population diversity. Results indicate that the TGA appears to have the highest level of difficulty in solving problems with a disordered pattern. While these problems seem to improve the CGA's performance, it has a negative affect on the performance of the TGA. However, the results illustrate that the TGA performs better on NK-like problems (i.e. the overlapped problems) and NK problems with higher levels of epistasis.

Categories and Subject Descriptors

H.4 [**EC**]: Evolutionary Computation

Keywords

Genetic Algorithms, Diversity, Epistasis

1. INTRODUCTION

Inspired by the evolution of living organisms, Genetic Algorithms (GAs) are one of the Evolutionary Computation (EC) algorithms which were introduced by John H. Holland [11]. Much research has been undertaken to illustrate that genetic algorithms are a useful approach for dealing with NP-Complete problems. However, despite the success of GAs in many domains and in many classes of problems, GAs still have difficulty with some problems. Two of the features associated with difficulty for GAs, deceptiveness and epistasis, have been the focus of much research in GA literature. Deceptive problems, for example [6, 26] have been shown to be difficult for GAs. These kinds of problems have been called GA-Hard Problems [6]. The local optima that lead the GA away from the global optimum in deceptive problems are called deceptive attractors [26].

[1]This work is based on an earlier work: SAC '12 Proceedings of the 2012 ACM Symposium on Applied Computing, Copyright 2012 ACM 978-1-4503-0857-1/12/03. http://doi.acm.org/10.1145/2245276.2245324.

Problems containing high levels of epistasis are also widely recognised as being difficult to optimise for GAs [4, 20, 18]. Epistasis can be viewed as an expression of the degree of linkage between genes in a chromosome [16]. The notion of epistasis in relation to GAs was introduced by Rawlins [19], where minimal epistasis exists when each gene is independent of every other gene and maximum epistasis relates to a situation where no gene is independent of any other gene. In general, when designing your genotype, it is often viewed that small changes in the object should also lead to minor changes in the behaviour of the object [25]. This is known as the principal of causality, as outlined in [17]. Both deceptive and epistatic problems violate this principal as small alterations of the genotype can lead to major changes in fitness. Maintaining a suitable balance between sufficient diversity and suitably efficient convergence is an ongoing important problem in GA research.

This paper compares the use of a Triploid GA (TGA) [14] which uses a multi-chromosome representations against that of a canonical GA (CGA) over a number of different deceptive and epistatic landscapes. The TGA uses a dominant chromosome to converge to optima in the space while a recessive chromosome is used to maintain and promote diversity. A third chromosome is also used to search for local minima near the current optima. This is to check whether the current fittest individual is a local optimum in order to reduce the time spent searching near this local optimum. These three chromosomes combine to assist the TGA in searching the landscape. The paper's structure is as follows: Section 2 examines some background concepts. Section 3 outlines the test suite used for the experiments, while Section 4 presents and analyses the experimental results. Finally, Section 5 presents conclusions.

2. BACKGROUND CONCEPTS

2.1 Diversity

Much research has focused on solving deceptive problems by maintaining population diversity [10, 3, 21, 1, 22, 23, 2]. One possible approach is to maintain a many-to-one relationship between genotype and phenotype using multi-layered genotype-phenotype models [10, 3, 21]. There are also several other explicit diversity maintenance methods such as using multi-chromosomes with dominance [1, 22], using multiple populations [23], and utilising reserve selection in the algorithm [2]. There is no doubt that increasing

diversity is one of the key approaches to solving deceptive problems. However, there are several factors that have a direct impact on GAs' performance in deceptive problems. In addition to the population diversity, convergence is another important property of genetic algorithms. We use the term convergence direction to describe the path to the building block to which the GA is currently being attracted. If the convergence direction repeatedly changes, the GAs will patrol between several solutions and will not fully converge to any of them. In general, increasing diversity will slow down the GA's convergence speed, and if the GA converges to one direction too fast, the GA will lose its diversity very quickly. One approach to maintaining diversity is to build a many-to-one mapping between genotype and phenotype. The ability to solve deceptive problems with this approach has been shown [10, 3, 21]. The inefficiency of this approach due to the lack of convergence has also been shown. In order to find a GA which can solve deceptive problems more accurately and efficiently, the balance between convergence and diversity must be found.

2.2 Deceptive Problems

Deceptive problems [6] are probably the best known GA-Hard problem, in which the solution space does not completely converge to the global optima. Some solutions converge to local optima in the solution space; the local optima that may attract the algorithm to converge are called deceptive attractors [26]. Manifestly, the difficulty of deceptive problems increases as the number of deceptive attractors and the percentage of the solutions that converge to the deceptive attractors increases. Furthermore, if the deceptive attractor has a very close fitness score to the global optimum, or, if it is more attractive, then the deceptive problems are more difficult. There are also deceptive problems defined in both the discrete space and continuous space. They have a common trait in that their solution spaces are non-monotonic. As one form of GA-hard problems, deceptive problems can be found in may areas from mathematics problem solving to fitness landscape searching. Many NP complete problems are deceptive, e.g. the traveling sales person problem. Therefore, it is very important for genetic algorithms to deal with problems which are deceptive. Many approaches can prevent GAs being trapped at deceptive attractors, as diversity plays an important role in solving deceptive problems.

2.2.1 Order-N Problem

The Order-N problem is a classic deceptive problem in discrete space, which was first introduced by Goldberg as a minimal deceptive problem [6]. The most used order-N problems are order-3 and order-4 problems.

2.2.2 Order-3 Problem

Due to the small solution space, the GAs' searching strategy can not obviously be seen in either order-3 or order-4 problems. Subsequently, Goldberg, Deb, and Korb defines a 10-dimensional order-3 problem [7], which each pattern of the problem can be presented as in Table 1.

2.2.3 Order-4 Problem

By increasing each pattern of the order-3 problem from 3 bits to 4 bits, Chow designed order-4 problems in his paper [3]. Order-4 problems have been divided into two categories, "Bad" order-4 problems and "Ugly" order-4 problems, which are described in Table 2.

Table 1: Order-3 problem

f(111)	30	f(101)	0
f(110)	0	f(011)	0
f(100)	14	f(010)	22
f(000)	28	f(001)	26

Table 2: "Bad" and "Ugly" Order-4 Problems

Order-4 Ugly				Order-4 Bad			
f2(1111)	30	f2(0110)	14	f3(1001)	30	f3(0101)	14
f2(0111)	0	f2(0101)	16	f3(1011)	0	f3(0011)	16
f2(1011)	2	f2(0011)	18	f3(1101)	2	f3(0000)	18
f2(1101)	4	f2(1000)	20	f3(1000)	4	f3(0111)	20
f2(1110)	6	f2(0100)	22	f3(0001)	6	f3(1110)	22
f2(1100)	8	f2(0010)	24	f3(1111)	8	f3(0100)	24
f2(1010)	10	f2(0001)	26	f3(1100)	10	f3(0010)	26
f2(1001)	12	f2(0000)	28	f3(1010)	12	f3(0110)	28

2.3 Epistatic Problems

In continuing to develop an understanding of the TGA, we examine another factor, epistasis, which is associated with making one problem harder than another to optimise for a GA. Epistasis can be viewed as the degree to which a gene is dependent upon other genes; in other words, minimal epistasis relates to a situation where each gene is independent for all other genes and maximum epistasis is where a genes is dependent on all other genes [16]. For GAs epistasis can be viewed as the extent of "*nonlinearity and interdependency among the elements composing the representation*" [5]. It has also been shown that deceptive problems cannot contain low epistasis and also problem functions with high epistasis are not always deceptive [16], but if a problem is deceptive, then epistasis can differentiate between type I and the more difficult type II deceptivity [8]. Where the higher level of difficulty is explained by the behaviour of epistasis [16]. So although a problem with a high level of epistasis may not be deceptive it can still remain difficult for a GA to solve. By examining the performance of the TGA over epistatic problems we will obtain a better understanding of its problem solving ability.

2.3.1 NK Landscape

Stuart Kauffman [13] devised the "NK fitness Landscape" model to explore the way that epistasis controls the "ruggedness" of a landscape. By specifying a fitness function which allowed the ruggedness to be tuned by a single parameter, the NK model allows the development of a landscape reflecting a specified level of epistasis. The properties of NK landscapes have been the focus of much research i.e. [13, 24]. The NK model is a stochastic method for generating a fitness function $F : 0, 1^N \rightarrow R$ binary strings $x \in 0, 1^N$, where the genotype x consists of N loci, with two possible alleles at each locus x_i . It has two basic components: a structure for gene interactions, and a way this structure is used to generate a fitness function for all possible genotypes. The gene

interaction structure is created as follows: the genotype's fitness is the average of N fitness components contributed by each locus. Each gene's fitness component F_i is determined by its own allele, x_i , and also the alleles at K other epistatic loci (therefore K must fall between 0 and $N-1$). These K other loci could be chosen in any number of ways from the N loci in the genotype.

The NK model was introduced by Kauffman to have a problem independent model for constructing fitness landscapes that can gradually be tuned from smooth to rugged. The main parameters of the model are N, the number of genes in the genotype, i.e. the length of the strings that form the points in the landscape, and K, the number of other genes that epistatically influence a particular gene (i.e., the fitness contribution of each gene is determined by the gene itself plus K other genes) [12]. K sets the level of epistasis by determining the dependence the partial fitness of a gene at location n has on the genes in a neighbourhood of K other locations. The neighbourhood may be at the K locations nearest to n in the genotype or a set of K locations randomly picked from anywhere on the genotype. Following this, a series of N lookup tables are then generated, one for each gene location in the genotype. Each table has 2^{K+1} random entries in the interval $(0, 1)$. The fitness, F_{NK}, of a particular genotype is calculated by the function:

$$F_{NK} = \frac{1}{N} \sum_{n=1}^{N} f(x)$$

where the partial fitness $f(n)$ is obtained from the nth lookup table using the values of the genes in location n and its neighbourhood as the lookup key [15]. To illustrate this we examine the calculation of $f(n)$ where $N = 8$ and $K = 2$. In this example our genotype and the neighbourhood on n (shaded area) are shown in Figure 1.

Figure 1: Genotype and Neighbourhood of n

1	0	0	0	1	1	0	0

To discover $f(n)$, where $n = 011$ we look up Table 3 and see that it is 0.274095.

Table 3: n^{th} Lookup Table

0	0	0	0.724367
0	0	1	0.123989
0	1	0	0.987432
1	0	0	0.432809
1	1	0	0.987234
1	0	1	0.349566
0	1	1	0.274095
1	1	1	0.521926

2.4 Triploid GA Representation

The new triploid genetic algorithm is designed to make search more efficient. The dominance of the triploid representation is decided by the phenotype. Elitism and immigration have also been used in this representation. The main idea of the TGA is to use the recessive chromosome to help the dominant chromosome converge in the early stages. Once the dominant chromosomes have converged, the recessive chromosomes help maintain diversity. An extra chromosome (termed a reverse chromosome) has been added in TGA, which is used to search the nearest local minimum. To maintain the minima, the reverse chromosome should converge to the local minima. Following every N generations (reverse generation), the reverse chromosome should crossover with the dominant chromosome in the same individual by a certain rate (reverse rate). With the reverse chromosomes, the TGA can solve completely deceptive problems efficiently.

A difficulty in creating the TGA is that the dominant chromosomes converge on the global optimum while the reverse chromosomes converge on the local minimum in the same individual; two different selection functions cannot be used in one population within normal crossover. The problem is circumvented using multi-parent crossover. Two individuals (b1, b2) are selected from the current generation, the better the dominant chromosome's fitness score, the more opportunity it has of being selected. Another two individuals (w1, w2) are selected using an alternative metrics: the worse the reverse chromosome's fitness score, the more opportunity it has of being selected. During crossover, the children's dominant chromosome is produced by b1 and b2's dominant chromosome, and the children's reverse chromosome is produced by w1 and w2's chromosomes. The children's recessive chromosome is produced by any two of the four recessive chromosomes.

2.5 Previous TGA Findings

The analysis of deceptive problems shows that the diversity is not the only parameter that may affect the GAs' performance in solving these kinds of problems. In fact, in completely non-deceptive problems, the GA does not have correct convergence to find the global optimum because it cannot maintain a definite convergence direction. Increasing diversity could help to solve the problems but it is not the only way to do it. Given enough generations, the canonical GA with elitism could solve deceptive problems. Therefore, a new approach to solving deceptive problems is by controlling the convergence direction has been proposed. The TGAs has been designed and tested over different problems in both discrete and continuous spaces [14]. The results show that increasing the diversity can increase the probability that GAs solve deceptive problems, and that the ability to maintain convergence directions affects the efficiency. Maintaining diversity while controlling the convergence direction is much more efficient than only maintaining the diversity [14].

3. TEST SUITE

3.1 2-bit overlapped order-4 problem

The 2-bit overlapped order-4 problem stems from the normal order-4 problem. However, it has two epistatic linkage genes situated between each two patterns (see Figure 2). Therefore, a 40 bit chromosome length will have 79 deceptive patterns.

The pattern's fitness mapping is shown in Table 4. In order

Figure 2: 2-bit overlapped order-4 100 bit problem

to illustrate the calculation of the best fitness levels for the 12 bit string shown in Table 2, we can see the four bits `1011` have a fitness value of 10, the next four bits `1001` have a fitness value of 26 and the last four bits `0111` have a value of 2. The 2-bit overlapped order-4 problem includes fitness values for the last 2-bits and first 2-bits of adjacent four bits (shaded in light gray). Therefore, between the first and second four bits (`1011` and `1001`) we have the bits `1110` with a fitness value of 30 and between the second and third four bits shaded in dark grey, (`1001` and `0111`) we have four bits `0101` with the fitness value of 18. This leaves us with a fitness value of 76 $(10 + 26 + 2 + 30 + 18)$ for the twelve bit string in Table 2.

Table 4: 2-bit overlapped order-4 problem

f2(1111)	20	f2(0110)	0
f2(0111)	2	f2(0101)	18
f2(1011)	10	f2(0011)	24
f2(1101)	22	f2(1000)	14
f2(1110)	30	f2(0100)	16
f2(1100)	4	f2(0010)	8
f2(1010)	12	f2(0001)	28
f2(1001)	26	f2(0000)	6

3.2 Cross Pattern Order-4 Problem

The Cross pattern Order-4 problem is a variation of the Normal Order-4 problem. However, it does not have overlapped bits between each pattern; instead it uses an extra mapping to allocate the genes from each pattern as outlined in Figure 3. Therefore, if we use the fitness values outlined in Table 4, the fitness value for the 16-bit string `1001 0101 1111` in Figure 3 is 56 (30+2+0+24).

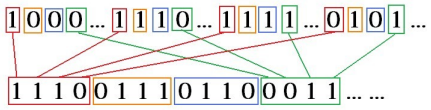


Figure 3: Cross pattern Order-4 problem

3.3 Disorder Order-4 Problem

We created a Disordered Pattern which is outlined in Figure 4. The Disordered order-4 problem is a variation of the order-4 problem. The disordered nature of the pattern makes the problem landscape more difficult to search rather than the ordered pattern, because of the defining length which increases the distance between building blocks and increases the probability of disruption when using crossover.

4. EXPERIMENTS

All of the experiments carried out in this paper were for 25,000 generations over 200 runs.

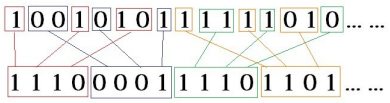


Figure 4: Disordered pattern Order-4 problem

4.1 2-bit Overlapped Order-4 Problem Experiments

The CGA and TGA both ran for 25,000 generations for the 2-bit overlapped order-4 problem. The experiments were conducted with various chromosome lengths (CL) and various population sizes (PS). In attempting to allow fair comparison between the CGA and TGA, we have kept the PS differences constant between both algorithms, that is, the population size of the CGA is always three times that of the TGA.

Looking at Figure 5, we can see the results for the 2-bit overlapped order-4 100 bit problem. The average best fitness per generation for 25000 generations, are plotted for the CGA with a PS of 300; the TGA with a PS of 100; the CGA with a PS of 150 and the TGA with a PS 0f 50. The first two rows of Table 5 show in row one, the CGA with a population size of 300 achieving an average best fitness score of 1164 and the TGA with a population size of 100, achieving an average best fitness score of 1165. While row two shows the best fitness result for the CGA and TGA with CL of 100 but with smaller population sizes (CGA PS of 150 and TGA PS of 50). The results indicate that there is little difference between the CGA and TGA in relation to average best fitness scores, with a CL of 100 even by varying the PS.

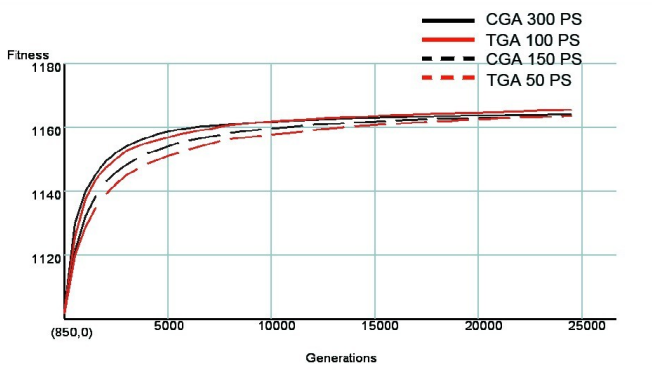


Figure 5: 2-bit overlapped order-4 100 bit problem

Figure 6 plots the performance of the CGA and TGA for a 2-bit overlapped order-4 problem using a CL of 60. Again the differences in the PS of both the CGA and TGA remain constant i.e., 300 for the CGA and 100 for the TGA. We can see that there is little difference between the performance of the CGA and the TGA using a CL of 60. Row three in Table 5 shows the average best fitness achieved of 649 for the CGA and 696 for the TGA, basically illustrating little or no difference between the two algorithms for their average best fitness score.

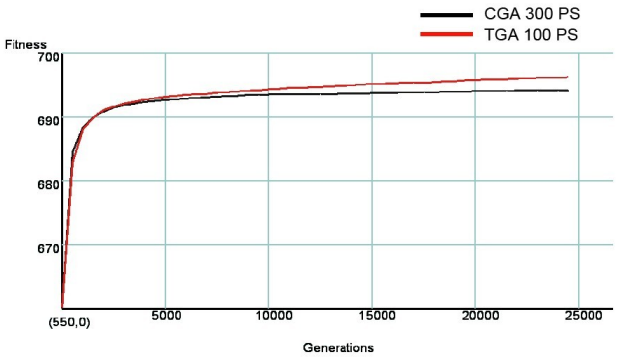


Figure 6: 2-bit overlapped order-4 60 bit problem

In Figure 7, we see the average best fitness for a 2-bit overlapped order-4 40 bit problem. This plot shows the results for the CGA and TGA with different pairs of PS i.e. CGA PS of 300 and TGA with PS of 100 and CGA with a PS of 150 and the TGA with a PS of 50. It is interesting that although there is little difference between the CGA and the TGA, they both perform similarly over the two different PS. Rows four and five in Table 5 show the average best fitness for the CGA with a PS of 300 at 458 and the average best fitness of 457 for a PS of 150. Although this is a small difference in terms of raw fitness score, it does mean that with a smaller population the CGA settles in a local optima and fails to find the better solution. Whereas the TGA has an average best fitness of 460 for both a PS of 100 and a PS of 50.

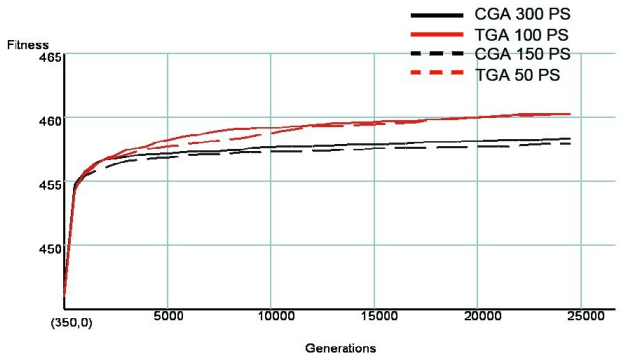


Figure 7: 2-bit overlapped order-4 40 bit problem

Figure 8 contains the average best fitness for both the CGA and the TGA with PS of 300 and 100 respectively, over a 2-bit overlapping order-4 30 bit problem. The results indicate the TGA, with a PS of 100, marginally outperforms and the CGA with a PS of 300. The average best performance as seen in row six of Table 5, shows the CGA achieving an average best fitness of 340 and the TGA achieving an average best fitness of 342. A plot of the results of the CGA with a PS of 300 and the TGA with a PS of 100 over a 2-bit overlapped order-4 20 bit problem is shown in Figure 9. Again we see little or no difference between the average best fitness of the CGA and the TGA.

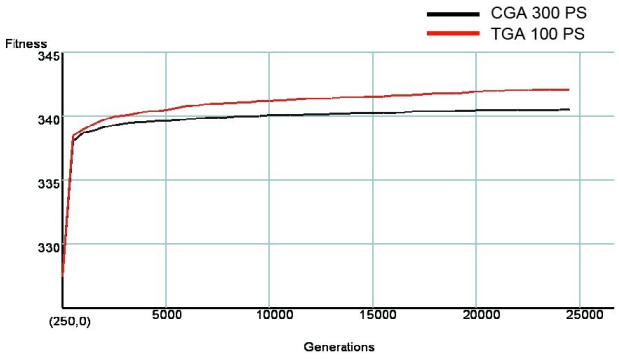


Figure 8: 2-bit overlapped order-4 30 bit problem

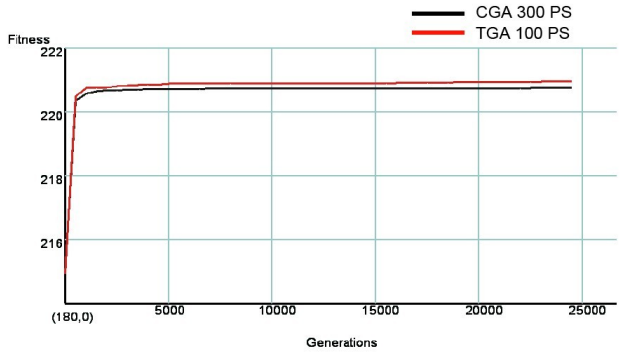


Figure 9: 2-bit overlapped order-4 20 bit problem

Table 5: 2-bit overlapped order-4 problem best fitness score

Row	Problem	CGA	TGA
1	100 CL, 300/100 PS	1164	1165
2	100 CL, 150/50 PS	1163	1163
3	60 CL, 300/100 PS	694	696
4	40 CL, 300/100 PS	458	460
5	40 CL, 150/50 PS	457	460
6	30 CL, 300/100 PS	340	342
7	20 CL, 300/100 PS	220	220

The experiment results outlined, show that the TGA has slightly better performance on 30/40/60 bit problems than the CGA, while both of them perform similarly on the 20/100 bit problems. This is probably because the 20 bit problem is much easier and the 100 bit problem is much more complicated. However, the figures (5, 6, 7, 8 and 9) shows the CGA converges very fast at the beginning, and after that, the plot goes flat, while the TGA still maintains the ability to find better solutions.

To gain a further understanding of the results we examined in more detail the performance of the CGA and the TGA

with a CL of 20. Following numerous runs the best fitness score obtained had a value of 222. In the following experiments we use a PS of 150 for the CGA and a PS of 50 for the TGA, to test how many times that CGA and TGA could obtain a fitness value of 222. The results of which are summarised in Table 6 and show that the CGA was successful 74 times out of 200 runs, while the TGA was successful 104 times out of 200. Also the results indicate that when the CGA appeared to locate the best fitness level, it was on average at generation 750 while the TGA, although locating it far more often, on average located it at generation 3086.

Table 6: 2-bit overlapped order-4 problem average generation of reach fitness score 222

Problem	CGA	TGA
Reach times	74/200	104/200
Avg. Gen	750	3086

The results indicate that the TGA located the optimum more often than the CGA. However, when the CGA did locate the optimum, it did so in fewer generations when compared with the TGA. One reason for this is that the CGA converges very quickly in the beginning, because the chromosome length is quite small. Furthermore, even though the CGA sometimes could find the solution very quickly, due to the diversity-maintaining mechanism associated with the TGA, the TGA has a better chance than the CGA to find better solutions.

From the experiments conducted we believe that the TGA is better than CGA because even though the TGA may only locate, on average, a marginally higher fitness score (as outlined in Figures 5, 6, 7, 8 and 9), it may indicate that the CGA is trapped in an deceptive attractor which may be quite far away from the optimum. Overall when comparing the CGA and the TGA over a normal order-4 overlapping problem, the TGA does not appear to have a significant advantage over the CGA. However due to the nature of the problem landscape, as explained in this section, we are uncertain as to what the global landscape is like. To try and shed more light onto the performance of the TGA, we conducted another series of experiments over a Cross pattern Order-4 problem landscape.

4.2 Cross Pattern Order-4 Problem Experiments

For the Cross Pattern Order-4 problem experiments we used an Ugly 40 and 60 bit chromosome and a Bad 40 and 60 bit chromosome [9] [3]. It is very interesting to note that in the cross pattern problems, the problem landscape appeared to impact more negatively on the TGA than on the CGA.

Table 7: Cross pattern Order-4

Problem	CGA(PS 300)	CGA(PS 100)	TGA(PS 100)
Ugly 40	2951 (200/200)	8897 (200/200)	7583 (199/200)
Ugly 60	19193 (57/200)	- (0 / 200)	17718 (1/200)
Bad 40	2931 (200/200)	7092 (200/200)	7751 (200/200)
Bad 60	19428 (86/200)	23925 (1/200)	- (0/200)

The TGA, in past research, has benefited from crossover (with the local minima, which are close to the global optimum). In this problem however, the crossover operation has a higher probability of disrupting building blocks due to the distance between building blocks caused by the increased defining length inherent in the problem landscape. Normally crossover will have an effect on almost every pattern, but in this situation the TGA cannot benefit from it's reverse chromosome anymore. However, the CGA still has the benefit of the population size as we use one third of the CGA population with the TGA. We feel that there are three reasons of this;

1. The TGA cannot benefit from the reverse chromosome anymore in this landscape.

2. The reordered landscape actually provide a good mechanism for preserving diversity, so the CGA also has a diversity maintaining mechanism to compete with the recessive chromosome in TGA.

3. The CGA has a larger population, therefore it has a proportionally greater chance of solving the problem. This may be one explanation as to why the TGA performance is far worse than that of the CGA over this problem. Once we reduce the population of CGA, it impacts negatively on the CGA and the performance is comparable with and in some cases worse, that that of the TGA.

One other item of interest emerging from these results is that even with more effective crossover it still does not guarantee the GA a better diversity rate [14]. We tested the diversity rate on CGA on both cross pattern and normal pattern 40 bit Ugly order-4 problems and the results are shown in Table 7:

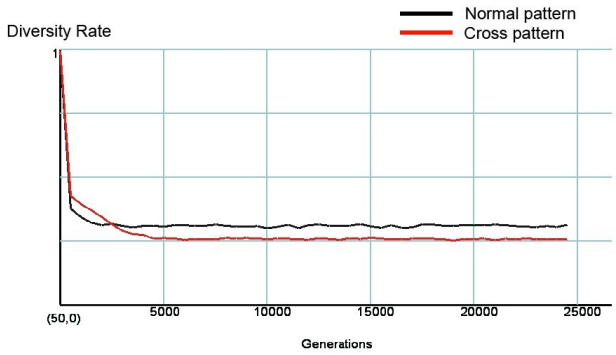


Figure 10: Diversity Rate plot

The result shows that the cross pattern experiments have higher diversity rates for $2,000-3,000$ generations and then the diversity rate drops to around 0.25. The canonical 40 bit ugly order-4 problem's diversity score goes down very quickly to around 0.29 at the beginning of the evolution, and continues to hover around that figure. Look at past experiment results [14], the CGA found the global optimum at cross pattern / canonical order-4 ugly problem at 2900/1500 generations respectively. The cross point of the diversity curves in in Figure 10 is around $2,000-3,000$ generations,

which means that the cross pattern will made the GA converge slower, but once converged, the diversity score will be even poorer than that of the canonical pattern.

4.3 Disorder Order-4 Experiments

To examine the significance of the effect on the TGA of crossover over the disordered crossed pattern problem (see Figure 4), we conducted another set of experiments which we tested on a 40 bit order-4 Ugly. The results of these experiments are summarised in Table 8.

Table 8: Disorder Order-4

Problem	CGA(PS 300)	TGA(PS 100)
Ugly 40	1360 (200/200)	1545 (199/200)

The disorder problem is quite similar to the cross pattern problem, but it has the effect of reducing the influence that crossover may have. Compared with the results of the cross pattern problem, the TGA appears to have a less negative effect than with the disordered problem. Taken together with the crossed pattern experiment, this may mean that crossover can have a significant influence on the performance of the TGA. Moreover, we have also tested the CGA and the TGA on a canonical 40 bit "ugly" order-4 deceptive problem, where the average global optimum was reached at generation 1549 (200/200) for CGA, and 585 (200/200) for TGA. We found that with the Disorder problem, the CGA may even find the global optimum (1360) a little faster than the normal pattern order-4 problem, while the TGA's convergence speed has slowed significantly. Through the result of the experiment on a normal ordered[2], crossed, and disorder landscape, we found that the reordered landscape may improve the performance of GAs over the disordered or not crossed problems, it is depended on the reorder mapping approach. Using a disorder approach in creating a genotype to phenotype mapping, has a slightly positive effect on the GAs, but it is not very effective and should only be noted when the disorder mapping is well designed. This result is similar to that discovered by Rick Chow [3], where they used a permutation operator to reset the order of phenotype bit and got a better result in solving deceptive problems.

4.4 NK Experiment Results

For the NK experiments we considered three different settings, one containing low epistasis, one containing a medium level of epistasis and one containing a high level of epistasis. The results are outlined in Figures 11, 12 and 13 below.
In Figure 11 we can see that there is little or no difference between the CGA and the TGA over the landscape generated by the NK model, where N=32 and K=5. Therefore, at low levels of epistasis, the TGA does not appear to offer any advantage over the CGA when searching the type of landscape generated by the NK model. When we examine an NK model landscape containing a medium level of epistasis i.e. where N=32 and K=15 as illustrated in Figure 12, there is no difference of significance between the performance of the CGA and the TGA, indicating that over this type of NK landscape containing a medium level of epistasis, there is no apparent advantage in using the TGA over the CGA. Finally, on an NK landscape containing a high level of epistasis, where N=32 and K=27, we can see in Figure 13 that the TGA outperforms the CGA. This indicates, bearing in mind the nature of the problem, that there appears to be an advantage in using the TGA over NK like problems where the levels of epistasis are relatively high. Our findings were shown to be statistically significant using a Wilcoxon rank sum test.

[2]normal/canonical order means that landscape is not reordered

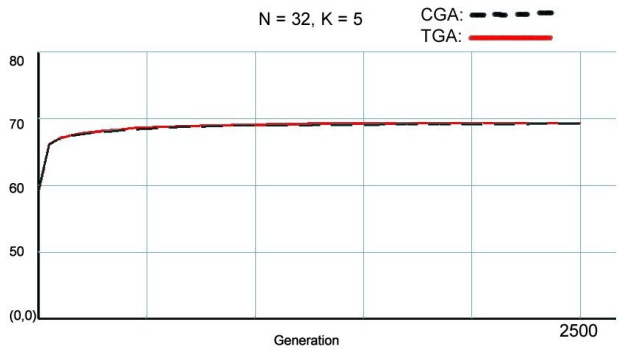


Figure 11: NK Model $N = 32$ and $K = 5$

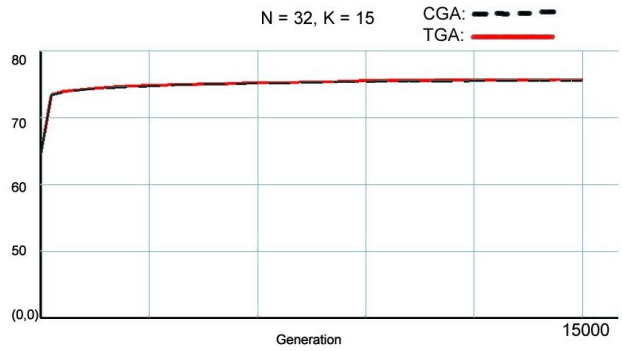


Figure 12: NK Model $N = 32$ and $K = 15$

5. CONCLUSION

From the experiments outlined above the results indicate that the TGA appears to have the highest level of difficulty in solving problems with a disordered pattern. While the disorder mapping seems to improve the canonical GA's performance, it has a negative affect on the TGA. However, bearing this in mind the TGA performs better on the NK like problems (i.e. the overlapped problems). More specifically, when we examined the performance of the TGA over NK landscapes with varying levels of epistasis, the TGA performed better at higher levels of epistasis. But it should also be noted that in our attempt to allow fair comparison, all the experiments conducted have a ratio of 3:1 for the population size. That is the population used by the CGA is consistently three times the population size of the TGA,

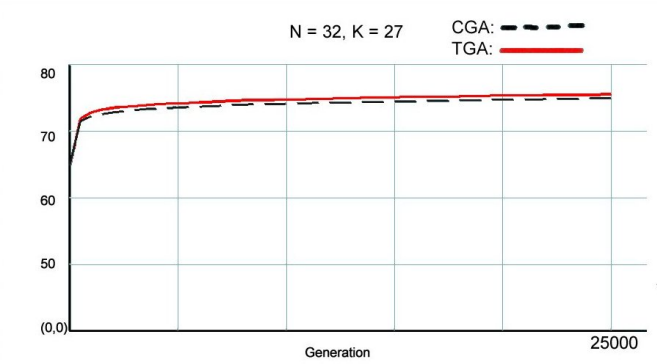


Figure 13: NK Model $N = 32$ and $K = 27$

which plays a critical role, as once we set the population sizes equal to one another, then the TGA always outperforms the CGA.

6. REFERENCES

[1] R. Cavill, S. Smith, and A. Tyrrell. Multi-chromosomal genetic programming. In *Proceedings of the 2005 conference on Genetic and evolutionary computation*, GECCO '05, pages 1753–1759, New York, NY, USA, 2005. ACM.

[2] Y. Chen, J. Hu, K. Hirasawa, and S. Yu. Gars: an improved genetic algorithm with reserve selection for global optimization. In *Proceedings of the 9th annual conference on Genetic and evolutionary computation*, GECCO '07, pages 1173–1178, New York, NY, USA, 2007. ACM.

[3] R. Chow. Evolving genotype to phenotype mappings with a multiple-chromosome genetic algorithm. In *Genetic and Evolutionary Computation - GECCO 2004*, volume 3102 of *Lecture Notes in Computer Science*, pages 1006–1017. Springer Berlin / Heidelberg, 2004. 10.1007/978-3-540-24854-5_100.

[4] Y. Davidor. Epistasis variance: A viewpoint on ga-hardness. In *FOGA*, pages 23–35, 1990.

[5] Y. Davidor. Epistasis variance: Suitability of a representation to genetic algorithms. *Complex Systems*, 4:369–383, 1990.

[6] D. Goldberg. Simple genetic algorithms and the minimal deceptive problem. *Genetic Algorithms and Simulated Annealing*, pages 74–88, 1987.

[7] D. Goldberg, K. Deb, and B. Korb. Messy genetic algorithms: motivation, analysis, and first results. *Complex Systems*, pages 493–530, 1989.

[8] D. E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley Publishing Company, Inc., Reading, MA, 1989.

[9] V. S. Gordon, V. S. Gordon, D. Whitley, and D. Whitley. Serial and parallel genetic algorithms as function optimizers. In *Proceedings of the Fifth International Conference on Genetic Algorithms*, pages 177–183. Morgan Kaufmann, 1993.

[10] S. Hill and C. O'Riordan. Solving fully deceptive problems in changing environments. In *21st conference on Artificial Intelligence and Cognitive Science (AICS)*, July 2010.

[11] J. H. Holland. Genetic algorithms. *Scientific American*, pages 66–72, July 1992.

[12] W. Hordijk. *Population flow on fitness landscapes*. PhD thesis, University of Rotterdam, 1994.

[13] S. Kauffman. *The Origins of Order*. Oxford University Press, 1995.

[14] M. Li, S. Hill, and C. O'Riordan. An analysis of multi-chromosome gas on deceptive problems. In *GECCO: Genetic and Evolutionary Computation Conference 2011*, 2011.

[15] G. Mayley. The evolutionary cost of learning. Technical report, School of Cognitive and Computer Sciences, University of Sussex, 1996.

[16] B. Naudts and A. Verschoren. Epistasis and deceptivity. In *Simon Stevin - Bulletin of the Belgian Mathematical Society , 6 , 147-154. Novkovic, S*, 1999.

[17] C. Palmer and A. Kershenbaum. Representing trees in genetic algorithms. In *Evolutionary Computation, 1994. IEEE World Congress on Computational Intelligence., Proceedings of the First IEEE Conference on*, volume 1, pages 379–384, jun 1994.

[18] M. Pelikan. Analysis of epistasis correlation on nk landscapes with nearest-neighbor interactions. In *Proceedings of the 13th annual conference on Genetic and evolutionary computation*, GECCO '11, pages 1013–1020, New York, NY, USA, 2011. ACM.

[19] G. J. E. Rawlins, editor. *Proceedings of the First Workshop on Foundations of Genetic Algorithms. Bloomington Campus, Indiana, USA, July 15-18 1990*. Morgan Kaufmann, 1991.

[20] C. R. Reeves and C. C. Wright. Epistasis in genetic algorithms: An experimental design perspective. In *Proc. of the 6th International Conference on Genetic Algorithms*, pages 217–224. Morgan Kaufmann, 1995.

[21] J. L. Risco-Martín, J. I. Hidalgo, J. Lanchares, and O. Garnica. Solving discrete deceptive problems with emmrs. In *Proceedings of the 10th annual conference on Genetic and evolutionary computation*, GECCO '08, pages 1139–1140, New York, NY, USA, 2008. ACM.

[22] R. E. Smith and D. E. Goldberg. Diploidy and dominance in artificial genetic search. *Complex Systems*, 6:251–285, 1992.

[23] P. T and R. K. R. A dual-population genetic algorithm for adaptive diversity control. *Evolutionary Computation, IEEE Transactions on*, Issue:99:1–1, June 2010.

[24] E. D. Weinberger. Local properties of kauffman's Nk model: A tunably rugged energy landscape. *Phys. Rev. A*, 44:6399–6413, Nov 1991.

[25] T. Weise, S. Niemczyk, H. Skubch, R. Reichle, and K. Geihs. A tunable model for multi-objective, epistatic, rugged, and neutral fitness landscapes. In *Proceedings of the 10th annual conference on Genetic and evolutionary computation*, GECCO '08, pages 795–802, New York, NY, USA, 2008. ACM.

[26] L. D. Whitley. Fundamental principles of deception in genetic search. In *Foundations of Genetic Algorithms*, pages 221–241. Morgan Kaufmann, 1991.

ABOUT THE AUTHORS:

Menglin Li is currently a postgraduate student of IT department, form the National University of Ireland, Glaway. He was graduated in Mathematics department at the year of 2008 from Capital Normal University, Beijing, China. His research interest includes Genetic Algorithms, and Evolutionary Game Theory.

Seamus Hill is currently a lecturer in Computer Science & Information Technology in the School of Engineering & Informatics at the National University of Ireland Galway. He has spent a number of years in industry and his main research interests include Artificial Intelligence and Evolutionary Computation.

Colm O'Riordan is a lecturer in Computer Science & Information Technology in the School of Engineering and Informatics at NUI, Galway. His main research interests are in the fields of Artificial Intelligence, Evolutionary Computation and Information Retrieval. He has published over a 100 papers in refereed journals and conferences. He has published papers in the domains of evolutionary computation and evolutionary game theory, artificial life and multi agent systems, information retrieval and computational intelligence in games.

SART: Speeding up Query Processing in Sensor Networks with an Autonomous Range Tree Structure

Spyros Sioutas
Department of Informatics,
Ionian University
49100 Corfu, Greece
sioutas@ionio.gr

Alexandros Panaretos
Department of Informatics,
Ionian University
49100 Corfu, Greece
alex@ionio.gr

Ioannis Karydis
Department of Informatics,
Ionian University
49100 Corfu, Greece
karydis@ionio.gr

Dimitrios Tsoumakos
Department of Informatics,
Ionian University
49100 Corfu, Greece
dtsouma@ionio.gr

Giannis Tzimas
Dept. of Applied Informatics in
Management and Economy,
Techn. Educ. Institute
30200 Messolonghi, Greece
tzimas@teimes.gr

Dimitrios Tsolis
Cult. Herit. Management and
New Technologies Dept.,
University of Western Greece
30100 Agrinio, Greece
dtsolis@upatras.gr

ABSTRACT

We consider the problem of constructing efficient P2P overlays for sensornets providing "Energy-Level Application and Services". In this context, assuming that a sensor is responsible for executing some program task but unfortunately it's energy-level is lower than a pre-defined threshold. Then, this sensor should be able to introduce a query to the whole system in order to discover efficiently another sensor with the desired energy level, in which the task overhead must be eventually forwarded. In this way, the "Life-Expectancy" of the whole network could be increased. Sensor nodes are mapped to peers based on their energy level. As the energy levels change, the sensor nodes would have to move from one peer to another and this operation is very crucial for the efficient scalability of the proposed system. Similarly, as the energy level of a sensor node becomes extremely low, that node may want to forward it's task to another node with the desired energy level. The method presented in [15] presents a novel P2P overlay for Energy Level discovery in a sensornet. However, this solution is not dynamic, since requires periodical restructuring. In particular, it is not able to support neither join of sensor_nodes with energy level out of the ranges supported by the existing p2p overlay nor leave of *empty* overlay_peers to which no sensor_nodes are currently associated. On this purpose and based on the efficient P2P method presented in [16], we design a dynamic P2P overlay for Energy Level discovery in a sensornet, the so-called SART (Sensors' Autonomous Range Tree) [1]. The adaptation of the P2P index presented in [16] guarantees the best-known dynamic query performance of the above operation. We experimentally verify this performance, via the D-P2P-Sim simulator [2].

Categories and Subject Descriptors

H.2 [**Database Management**]: [Emergent Systems]; D.2 [**Software Engineering**]: [P2P Simulators for Sensornets, QoS]

General Terms

Distributed Data Structures, Indexing

Keywords

Peer-to-Peer Overlays, Sensor Networks

1. INTRODUCTION

In the last years sensornet research primarily focused on data collection, finding applications in ecology (e.g., environmental and habitat monitoring [13]), in precision agriculture (e.g., monitoring of temperature and humidity), in civil engineering (e.g., monitoring stress levels of buildings under earthquake simulations), in military and surveillance (e.g., tracking of an intruder [7]), in aerospace industry (e.g., fairing of cargo in a rocket), etc.

Traditionally, sensors are used as data gathering instruments, which continuously feed a central base station database. The queries are executed in this centralized base station database which continuously collates the data. However, given the current trends (increase in numbers of sensors, together collecting gigabits of data, increase in processing power at sensors) it is not anymore feasible to use a centralized solution for querying the sensor networks. Therefore, there is a need for establishing an efficient access structure on sensor networks in order to contact only the relevant nodes for the execution of a query and hence achieve minimal energy consumption, minimal response time, and an accurate response. We achieve these goals with our peer-to-peer query processing model on top of a distributed index structure on wireless sensor networks.

In sensor networks any node should be able to introduce a query to the system. For example, in the context of a fire evacuation scenario a firefighter should be able to query a nearby sensor node for the closest exit where safe paths exist. Therefore, a peer-to-peer query processing model is required. A first P2P program for spatial query execution

[1]This work is based on an earlier work: SAC '12 Proceedings of the 2012 ACM Symposium on Applied Computing, Copyright 2012 ACM 978-1-4503-0857-1/12/03. http://doi.acm.org/10.1145/2245276.2245442.

[2]D-P2P-Sim is publicly available at http://code.google.com/p/d-p2p-sim/

presented in [8].

According to [1], the benefits of the P2P overlays in sensornets are the following: Efficient Data Lookup, Guaranties on Lookup Times, Location Independence, Overlay Applications and Services, Elimination of proxies/sinks with undesirable central authority, Limited Broadcast. P2P design, for Internet-like environments, has been a very active research area and there are many P2P Internet protocols and systems available like CAN [3], Pastry [3], and Chord [3]. The main arguments against P2P designs in sensornets were the following: Logical Topology=Physical Topology, Route Maintenance Overhead, Sensor Nodes are Not Named, DHTs are Computationally Intensive. By overcoming the arguments above (for details see [1], [2] and [4]), in [2] and [4] the first DHT (Distributed Hash Table) based protocols for sensornets were presented, the CSN and VRR respectively. In [1] the Tiered Chord (TChord) protocol was proposed, which is similar to, and inspired by, CSN. TChord is a simplified mapping of Chord onto sensornets. Unlike CSN the design of TChord is more generic (to support a variety of applications and services on top instead of just serving incoming data queries). Gerla et al. argue for the applicability and transfer of wired P2P models and techniques to MANETs [9].

Most existing decentralized discovery solutions in practice are either DHT based, like Chord or hierarchical clustering based, like BATON [3], NBDT [14], ART [16] or Skip-Graphs [3]. The majority of existing P2P overlays for sensornets were designed in a DHT fashion and the best current solution is the TChord. On the contrary, ELDT [15] is the only existing P2P protocol for sensornets, which combines the benefits of both DHT and hierarchical [14] clustering fashions. In this solution, sensor_nodes are mapped to peers based on their energy level. As the energy levels change, the sensor nodes would have to move from one peer to another and this oparation is very crucial for the efficient scalability of the proposed system. Similarly, as the energy level of a sensor node becomes extremely low, that node may want to forward it's task to another node with the desired energy level. However, the ELDT solution is not dynamic, since requires periodical restructuring. In particular, it is not able to support neither join of sensor_nodes with energy level out of the ranges supported by the existing p2p overlay nor leave of *empty* overlay_peers to which no sensor_nodes are currently associated. On this purpose and based on the efficient P2P method presented in [16], we design a dynamic P2P overlay for Energy Level discovery in a sensornet, the so-called SART (Sensors' Autonomous Range Tree). The adaptation of the P2P index presented in [16] guarantees the best-known dynamic query performance of the above operation.

The main functionalities of SART attempt to increase the "Life-Expectancy" of the whole sensor network in dynamic way, providing support for processing: (a) exact match queries of the form "given a sensor node with low energy-level k', locate a sensor node with high energy-level k, where $k >> k'$" (the task will be forwarded to the detected sensor node) (b) range queries of the form "given an energy-level range $[k, k']$, locate the sensor node/nodes the energy-levels of which belong to this range" (the task will be forwarded to one of the detected sensor nodes) (c) update queries of the form "find the new overlay-peer to which the sensor node must be moved (or associated) according to it's current energy level" (the energy level of each sensor node is a decreasing function of time and utilization) (d) join queries of the form "join a new overlay-peer to which the new (inserted) sensor node is associated" and (e) leave queries of the form "leave (delete) the overlay-peer to which no sensor nodes are currently associated". The SART overlay adapts the novel idea of ART P2P infrastructure presented in [16] providing functionalities in optimal time. For comparison purposes, an elementary operation's evaluation is presented in table 1 between ART, NBDT, Skip-Graphs [3], Chord [3] and its newest variation (F-Chord(á) [3]), BATON and its newest variation (BATON* [3]).

The rest of this paper is structured as follows. Section 2 and 3 describe the SART system while section 4 presents an extended experimental verification via an appropriate simulator we have designed for this purpose. Section 5 concludes the work.

2. THE SART PROTOCOL

SART, is a simplified mapping of ART [16] onto sensornets. Like ART, at the heart of SART, lookup and join/leave respectively are the two main operations. Given a set of sensor nodes, we hash the unique address of each sensor node to obtain node identifiers. Meta-data keys, generated from the data stored on the nodes, are hashed to obtain key identifiers.

The SART protocol (see figure 1) is an hierarchical arrangement of some sensor nodes (master nodes). The master node of level i maintains information (in its local finger table) about all its *slave nodes* and $2^{2^{i-1}}$ other master nodes (you can find more details about master and slave nodes in [15]). All queries are resolved in a distributed manner with a bound of $O(\log_b^2 \log N)$ messages. When a master node receives a query it first checks its own keys to resolve the query, if the lookup is not successful the master node then checks its local finger table. The finger table contains information about $2^{2^{i-1}}$ other master nodes and if the key can be located according to the information stored in the finger table, the query is directly forwarded to the master node storing the data. If the lookup on the local finger table also fails then the master node routes the query to the master node closest to the target according to the finger table. We handle the master_node joins/leaves and fails according to join/leave and fail operations respectively presented in [16]. Thus, all the above operations are bounded by $O(\log \log N)$ expected w.h.p. number of messages. Slave nodes do not store information about their neighbors. If a slave node directly receives a query, it checks its own data and if the lookup fails it simply forwards the query to its master node. For simplicity, in the SART proposal we opt for not connecting the slave nodes in a ART arrangement and lookups are not implemented in slave nodes. The master nodes could be thought as "virtual sinks" with an ART overlay between these virtual sinks. Unlike IP in the Internet, the sensornet protocol SP is not at the network layer but instead sits between the network and data-link layer (because data-processing potentially occurs at each hop, not just at end points). Figure 2 shows

P2P Architectures	Lookup/update key	Data Overhead-Routing information	Join/Depart Node
Chord	$O(logN)$	$O(logN)$ nodes	$O(logN)$ w.h.p.
H-F-Chord(a)	$O(logN/loglogN)$	$O(logN)$ nodes	$O(logN)$
LPRS-Chord	$O(logN)$	$O(logN)$ nodes	$O(logN)$
Skip Graphs	$O(logN)$	$O(1)$	$O(logN)$ amortized
BATON	$O(logN)$	Two (2) nodes	$O(logN)$ w.h.p.
BATON*	$O(log_m N)$	m nodes	$O(mlog_m N)$
NBDT	$O(loglogN)$	$O(loglogN)$ or $2^{2^{i-1}}$ for nodes at level i of left spine	periodical restructuring
ART	$O(\log_b^2 \log N)$	$O(N^{1/4}/\log^c N)$ nodes	$O(\log \log N)$ expected w.h.p.

Table 1: Performance Comparison between ART, NBDT, Chord, BATON and Skip Graphs

how P2P overlays can be implemented on top of SP. The P2P overlay (shown as P2P Overlay Management) could be built on top of any generic network protocol. An underlying DHT or Hierarchical Clustering routing protocol (e.g., VRR, CSN, TChord or SNBDT or SART) is recommended as it simplifies the job of overlay management. In particular, it is more efficient to build routing directly on top of the link layer instead of implementing it as an overlay on top of a routing protocol [4]. P2P Services and Applications (e.g. event notification, resource allocation, and file systems) can then be built on top of the P2P overlay and sensornet applications could either use these services or communicate with the P2P overlay themselves.

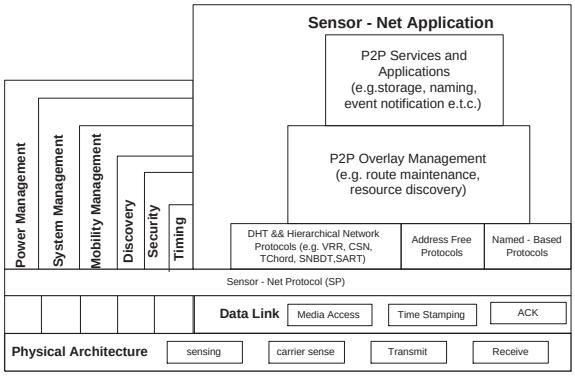


Figure 2: P2P Overlay in SP Architecture

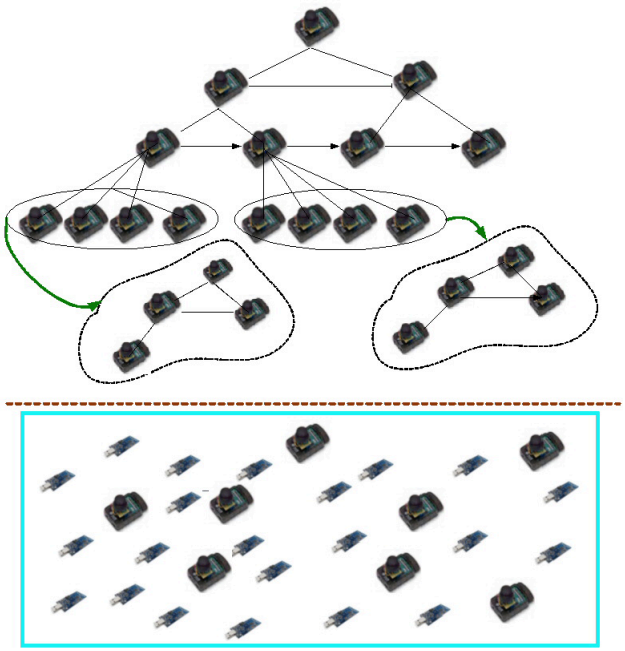


Figure 1: The SART protocol

3. THE SART P2P OVERLAY

Let G a network graph of n sensor nodes and SART the respective overlay of N peers. With each overlay peer p $(1 \leq p \leq N)$ we associate a set of pairs $S_p = \{(g, L[g])\}$, where g is a sensor node $(1 \leq g \leq n)$ and $L[g]$ its current energy level. The criterion of associating the sensor node g to peer p depends on it's current energy level. Obviously, it holds that $N << n$. Let's explain more the way we structure our whole system.

One of the basic components of the final SART structure is the LRT (**L**evel **R**ange **T**ree) [16] structure. LRT will be called upon to organize collections of peers at each level of SART.

3.1 The LRT structure: An overview

LRT [16] is built by grouping nodes having the same ancestor and organizing them in a tree structure recursively. The innermost level of nesting (recursion) will be characterized by having a tree in which no more than b nodes share the same direct ancestor, where b is a double-exponentially power of two (e.g. 2,4,16,...). Thus, multiple independent trees are imposed on the collection of nodes. Figure 3 illustrates a simple example, where $b = 2$.

The degree of the overlay peers at level $i > 0$ is $d(i) = t(i)$, where $t(i)$ indicates the number of peers at level i. It holds that $d(0)$=2 and $t(0)$=1. Let n be w-bit keys. Each peer with label i (where $1 \leq i \leq N$) stores ordered keys that belong in the range $[(i-1)\ln n, i\ln n–1]$, where $N = n/lnn$ is the number of peers. Each peer is also equipped with a table named *Left Spine Index* (LSI), which stores pointers to the peers of the left-most spine (see pointers starting from

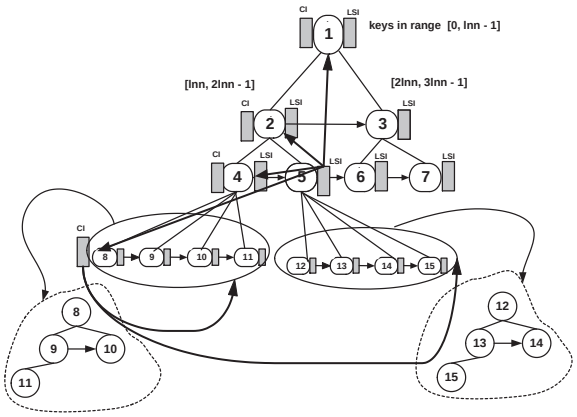


Figure 3: The LRT structure

peer 5). Furthermore, each peer of the left-most spine is equipped with a table named *Collection Index* (CI), which stores pointers to the collections of peers presented at the same level (see pointers directed to collections of last level). Peers having the same father belong to the same collection.

Lookup Algorithm: Assume we are located at peer s and seek a key k. First, the algorithm finds the range where k belongs. If $k \in [(j-1)\ \ln n, j \ln n - 1]$, it has to search for peer j. The first step of algorithm is to find the LRT level where the desired peer j is located. For this purpose, it exploits a nice arithmetic property of LRT. This property says that for each peer x located at the left-most spine of level i, the following formula holds:

$$label(x) = label(father(x)) + 2^{2^{i-2}} \quad (1)$$

For each level i (where $0 \leq i \leq \log\log N$), it computes the value x of its left most peer by applying Equation (1). Then, it compares the value j with the computed value x. If $j \geq x$, it continues by applying Equation (1), otherwise it stops the loop process with current value i. The latter means that node j is located at the i-th level. Then, it follows the i-th pointer of the LSI table located at peer s. Let x the destination peer, that is the leftmost peer of level i. Now, the algorithm must compute the collection in which the peer j belongs to. Since the number of collections at level i equals the number of nodes located at level $(i-1)$, it divides the distance between j and x by the factor $t(i-1)$ and let m the result of this division. Then, it follows the $(m+1)$-th pointer of the CI table. Since the collection indicated by the CI[m+1] pointer is organized in the same way at the next nesting level, it continues this process recursively.

Analysis: Since $t(i) = t(i-1)d(i-1)$, it gets $d\,(i) = t\,(i) = 2^{2^{i-1}}$ for $i \geq 1$. Thus, the height and the maximum number of possible nestings is $O(\log\log N)$ and $O(\log_b \log N)$ respectively. Thus, each key is stored in $O(\log_b \log N)$ levels at most and the whole searching process requires $O(\log_b \log N)$ hops. Moreover, the maximum size of the CI and RSI tables is $O(\sqrt{N})$ and $O(\log\log N)$ in worst-case respectively.

Each overlay peer stores tuples $(g, L[g])$, where $L[g]$ is a $k-bit$ key belonging in universe $K = [0, 2^k - 1]$, which represents the current energy-level of the sensor node g. We associate to i^{th} peer the set $S_i = \{(g, L[g])\}$, where $L_g \in [(i-1)lnK, ilnK - 1]$. Obviously, the number of peers is $N = K/lnK$ and the load of each peer becomes $\Theta(polylogN)$ in expected case with high probability (for more details see[1]). Each energy-level key is stored at most in $O(loglogN)$ levels. We also equip each peer with the table LSI (Left Spine Index). This table stores pointers to the peers of the left-most spine (for example in figure 3 the peers 1, 2, 4 and 8 are pointed by the LSI table of peer 5) and as a consequence its maximum length is $O(loglogN)$. Furthermore, each peer of the left-most spine is equipped with the table CI (Collection Index). CI stores pointers to the collections of peers presented at the same level (see in figure 3 the CI table of peer 8). Peers having same father belong to the same collection. For example in the figure 2, peers 8,9,10 and 11 constitute a collection of peers. It's obvious that the maximum length of CI table is $O(\sqrt{N})$.

3.2 The ART structure: An Overview

The backbone of ART [16] is exactly the same with LRT. During the initialization step the algorithm chooses as cluster_peer representatives the 1st peer, the $(\ln n)$-th peer, the $(2\ln n)$-th peer and so on.

This means that each cluster_peer with label i' (where $1 \leq i' \leq N'$) stores ordered peers with energy-level keys belonging in the range $[(i'-1)\ln^2 n, \ldots, i' \ln^2 n - 1]$, where $N' = n/\ln^2 n$ is the number of cluster_peers.

ART stores cluster_peers only, each of which is structured as an independent decentralized architecture. Moreover, instead of the **L**eft-most **S**pine **I**ndex (LSI), which reduces the robustness of the whole system, ART introduces the **R**andom **S**pine **I**ndex (RSI) routing table, which stores pointers to randomly chosen (and not specific) cluster_peers (see pointers starting from peer 3). In addition, instead of using fat CI tables, the appropriate collection of cluster_peers can be accessed by using a 2-level LRT structure.

Load Balancing: The join/leave of peers inside a cluster_peer were modeled as the combinatorial game of bins and balls presented in [12]. In this way, for a $\mu(\cdot)$ random sequence of join/leave peer operations, the load of each cluster_peer never exceeds $\Theta(\text{polylog } N')$ size and never becomes zero in expected w.h.p. case.

Routing Overhead: The 2-level LRT is an LRT structure over $\log^{2c} Z$ buckets each of which organizes $\frac{Z}{\log^{2c} Z}$ collections in a LRT manner, where Z is the number of collections at current level and c is a big positive constant. As a consequence, the routing information overhead becomes $O(N^{1/4}/\log^c N)$ in the worst case (even for an extremely large number of peers, let say N=1.000.000.000, the routing data overhead becomes 6 for $c = 1$).

Lookup Algorithms: Since the maximum number of nesting levels is $O(\log_b \log N)$ and at each nesting level i the standard LRT structure has to be applied in $N^{1/2^i}$ collections, the whole searching process requires $O(\log_b^2 \log N)$ hops. Then, the target peer can be located by searching the respective decentralized structure. Through the polylogarithmic load of each cluster_peer, the total query com-

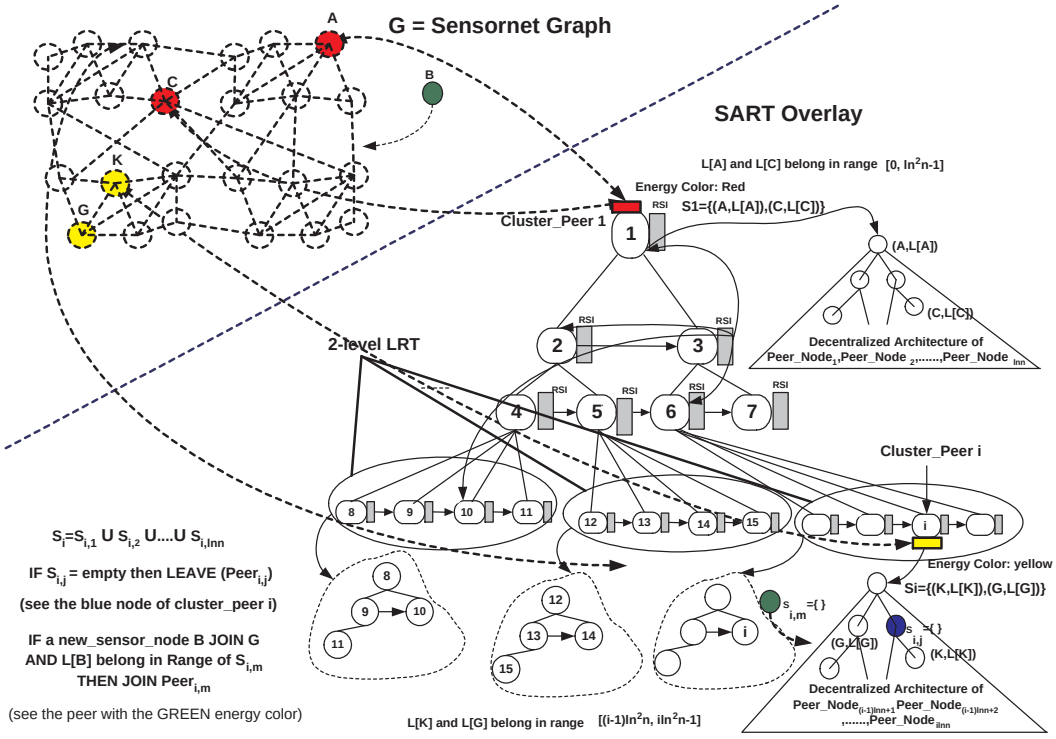


Figure 4: Building the SART Bipartite P2P Overlay

plexity $O(\log_b^2 \log N)$ follows. Exploiting now the order of keys on each peer, range queries require $O(\log_b^2 \log N + |A|)$ hops, where $|A|$ the answer size.

Join/Leave Operations: A peer u can make a join/leave request at a particular peer v, which is located at cluster_peer W. Since the size of W is bounded by a $polylogN$ size in expected w.h.p. case, the peer join/leave can be carried out in $O(loglogN)$ hops.

Node Failures and Network Restructuring: Obviously, node failure and network restructuring operations are according to the decentralized architecture used in each cluster_peer.

3.3 Building the SART Overlay

Let $P_{i,j}$ the j^{th} peer of cluster_peer i. Each overlay peer $P_{i,j}$, stores a set $S_{i,j} = \{(g, L[g])\}$, where $L[g]$ is a $k - bit$ key belonging in universe $K = [0, 2^k - 1]$, which represents the current energy-level of the sensor node g. In particular (and based on design analysis of previous section) it holds that $L[g] \in \left[(i-1)ln^2n, iln^2n - 1\right]$. Thus, the total set of $Cluster\_Peer\ i$ becomes $S_i = S_{i,1} \cup S_{i,2} \cup \ldots \cup S_{i,\Theta(polylogN)}$, where $|S_{i,j}| \leq n$.

For example in Figure 4, $S_1 = \{(A, L[A]), (C, L[C])\}$ is the set of cluster_peer 1, which stores the energy-level keys of red (energy color) sensors A and C as well as $S_i = \{(K, L[K]), (G, L[G])\}$ is the set of cluster_peer i, which stores the energy-level keys of yellow sensors K and G. Tuples $(A, L[A])$ and $(C, L[C])$ are located in different peers of the decentralized structure associated to cluster_peer 1. The same holds for the tuples $(K, L[K])$ and $(G, L[G])$ in the decentralized structure associated to cluster_peer i.

According to the complexity analysis of ART structure, the theorem 1 follows:

Theorem 1: Assume a $SART$ lookup $P2P$ system for the sensor network G. The queries of the form (a), (b) and (c) require $O(\log_b^2 \log N)$ expected w.h.p. number of messages. The queries of the form (d) and (e) require $O(\log \log N)$ expected w.h.p. number of messages.

Let G the sensor network and T the $SART$ overlay. We are located at sensor node $S \in G$ with low energy level k' and we are looking for a sensor node $R \in G$ with the desired energy level k. Algorithm 1 depicts the pseudocode for the Sensor_Net_Exact_Match_Search routine.

Let G the sensor network and T the $SART$ overlay. We are located at sensor node $S \in G$ with low energy level k' and we are looking for a sensor node $R \in G$ the desired energy level of which belongs in the range $[k_1, k_2]$. Algorithm 2 depicts the pseudocode for the Sensor_Net_Range_Search routine.

Let G the sensor network and T the overlay structure. We are located at sensor node $S \in G$, the energy level of which has been decreased from $k1$ to $k2$. We have to find the new overlay peer to which the update node S is going to be associated. Algorithm 3 depicts the pseudocode for the update_overlay_peer routine.

Let G the sensor network and T the overlay structure. If a new sensor node B joins G and $L[B] \in S_{i,m}$ then JOIN $P_{i,m}$ (see the peer with the green energy color). Algorithm 4 depicts the respective pseudocode.

Let G the sensor network and T the overlay structure. If $S_{i,j} = \oslash$ then LEAVE $P_{i,j}$ (see the blue node of cluster peer i). Algorithm 5 depicts the respective pseudocode.

Algorithm 1 Sensor_Net_Exact_Match_Search(G,S,T,k',k, R)

1: Find the peer node to which sensor S (of energy level k') is associated;
2: Let $p \in T$ the respective overlay peer;
3: $r = send\_overlay\_search(T, p, k)$; {it is the basic lookup routine of ART structure T}
4: Let $r \in T$ the peer node which stores sensor nodes with the desired energy-level k and let say R a randomly chosen one;
5: Return R

Algorithm 2 Sensor_Net_Range_Search(G,S,T,k',k_1,k_2,R)

1: Find the peer to which sensor S (of energy level k') is associated;
2: Let $p \in T$ the respective overlay peer;
3: $r = send\_overlay\_range\_search(T, p, k)$; {it is the range searching routine of ART structure T}
4: Let A the set of peers the desired energy-level of which belong in range $[k_1, k_2]$ and let say R a randomly chossen one;
5: Return R

Algorithm 3 Update_Overlay_Peer(G,T,S,k_1,k_2)

1: Find the peer to which S is associated according to old energy level k_1;
2: Let $p \in T$ the respective overlay peer;
3: Delete (S, k_1) from p;
4: $r = send\_overlay\_search(T, p, k_2)$;
5: Insert the tuple (S, k_2) into r;

4. EXPERIMENTS

The Admin tools of D-P2P-Sim GUI (see Figure 5) have specifically been designed to support *reports* on a collection of wide variety of metrics including, protocol operation metrics, network balancing metrics, and even server metrics. Such metrics include frequency, maximum, minimum and average of: number of hops for all basic operations (lookup-insertion-deletion path length), number of messages per node peer (hotpoint-bottleneck detection), routing table length (routing size per node-peer) and additionally detection of network isolation (graph separation). All metrics can tested using a number of different distributions (e.g. normal, weibull, beta, uniform etc). Additionally, at a system level memory can also be managed in order to execute at low or larger volumes and furthermore execution time can also be logged. The framework is open for the protocol designer to introduce additional metrics if needed. Furthermore, XML rule based *configuration* is supported in order to form a large number of different protocol testing scenarios. It is possible to configure and schedule at once a single or multiple experimental scenarios with different number of protocol networks (number of nodes) at a single PC or multiple PCs and

Algorithm 4 Join_Overlay_Peer(G,T,B,$L[B]$)

1: Let $L[B] \in S_{i,m}$ and the m^{th} peer of cluster_peer i does not exist;
2: $send\_join\_peer(T, P_{i,m})$; {it is the Join routine of ART structure T}
3: Let $S_{i,m} = \oslash$ the initial empty set of the new inserted peer $P_{i,m}$;
4: Insert the tuple $(B, L[B])$ into $S_{i,m}$;

Algorithm 5 Leave_Overlay_Peer(G,T,$P_{i,j}$)

1: Let $S_{i,j} = \oslash$ the empty set of peer $P_{i,j}$;
2: $send\_Leave\_peer(T, P_{i,j})$; {it is the Leave routine of ART structure T}

servers distributedly. In particular, when D-P2P-Sim simulator acts in a distributed environment (see Figure 6) with multiple computer systems with network connection delivers multiple times the former population of cluster peers with only 10% overhead.

Our experimental performance studies include a detailed performance comparison with TChord, one of the state-of-the-art P2P overlays for sensor networks. Moreover, we implemented each cluster_peer as a BATON* [10], the best known decentralized tree-architecture. We tested the network with different numbers of peers ranging up to 500,000. A number of data equal to the network size multiplied by 2000, which are numbers from the universe [1..1,000,000,000] are inserted to the network in batches. The synthetic data (numbers) from this universe were produced by the following distributions: beta[3], uniform[4] and power-law[5]. The distribution parameters can be easily defined in configuration file[6]. Also, the predefined values of these parameters are depicted in the figure 7.

For evaluation purposes we used the Distributed Java D-P2P-Sim simulator presented in [16]. The D-P2P-Sim simulator is extremely efficient delivering $> 100,000$ cluster peers in a single computer system, using 32-bit JVM 1.6 and 1.5 GB RAM and full D-P2P-Sim GUI support. When 64-bit JVM 1.6 and 5 RAM is utilized the D-P2P-Sim simulator delivers $> 500,000$ cluster peers and full D-P2P-Sim GUI support in a single computer system.

For each test, 1,000 exact match queries and 1,000 range queries are executed, and the average costs of operations are taken. Searched ranges are created randomly by getting the whole range of values divided by the total number of peers multiplies α, where $\alpha \in [1..10]$. The source code of the whole evaluation process is publicly available [7].

In the first tab (see Figure 8) the user can set the number of peers which will constitute the overlay and select the energy

[3]http://goo.gl/ltQXY
[4]http://goo.gl/Y1fEB
[5]http://goo.gl/lqp91
[6]http://goo.gl/dHZ6D
[7]`http://code.google.com/p/d-p2p-sim/`

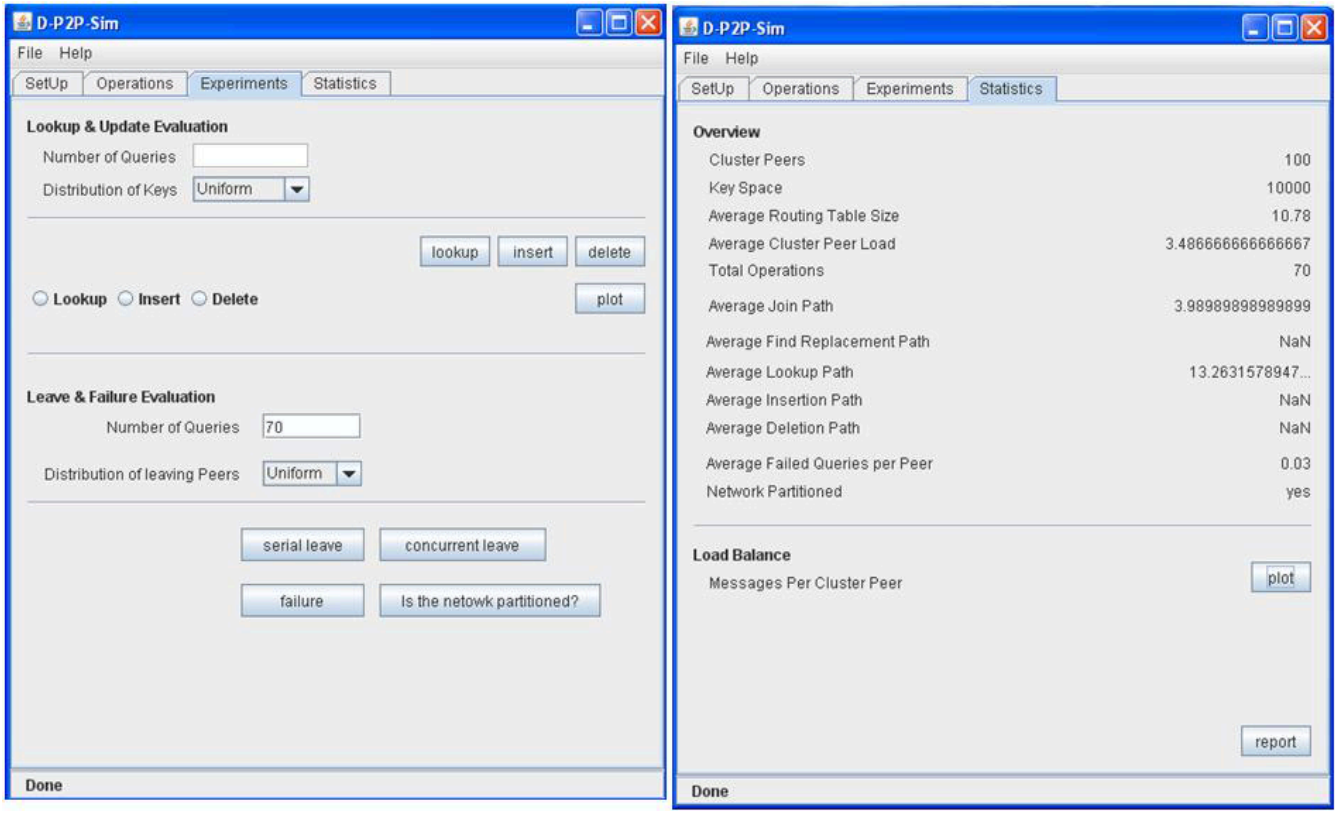


Figure 5: D-P2P-Sim GUI

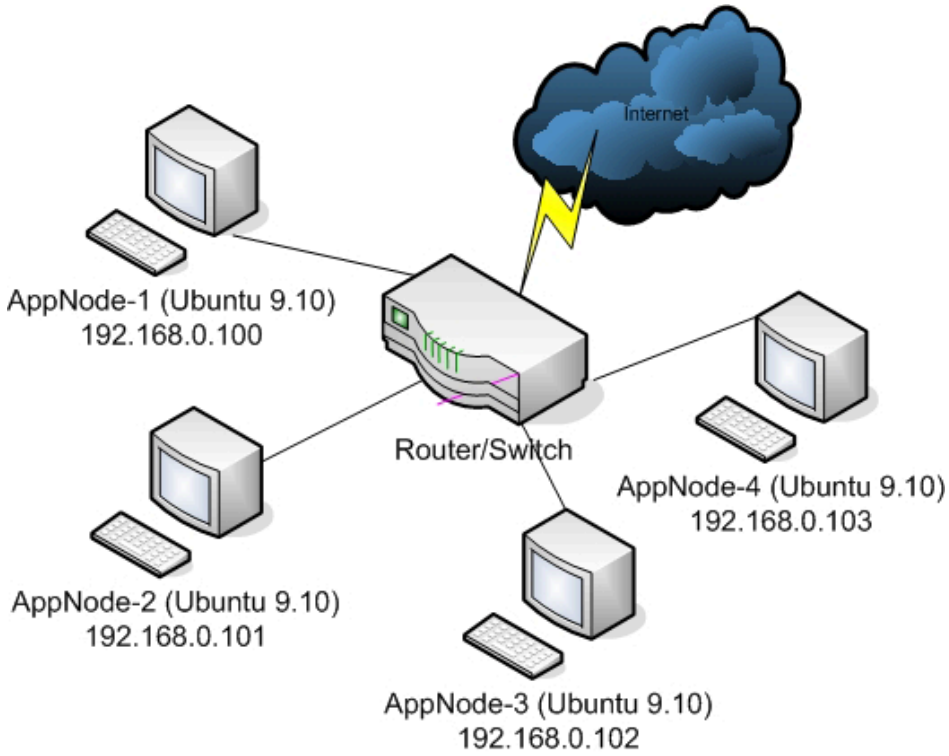


Figure 6: The Distributed Environment

```
<distribution>
  <random>
    <seed>1</seed>
  </random>
  <beta>
    <alpha>2.0</alpha>
    <beta>4.0</beta>
  </beta>
  <powerLaw>
    <alpha>0.5</alpha>
    <beta>1.0</beta>
  </powerLaw>
</distribution>
```

Figure 7: Snippet from config.xml with the predefined distribution's parameters setup

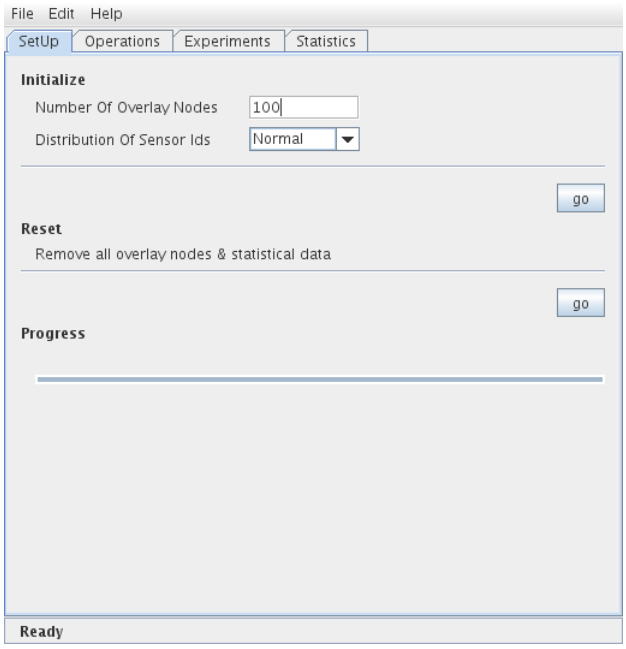


Figure 8: The tab "SetUp"

level distribution over these nodes. The available distributions are: uniform, normal, beta, and pow-law. After the user has set these two fields then the system's initialization can begin.

In the same tab there is a progress bar so the user can obtain the overall process due to the fact that this process may take several minutes. Also there is a button, which resets

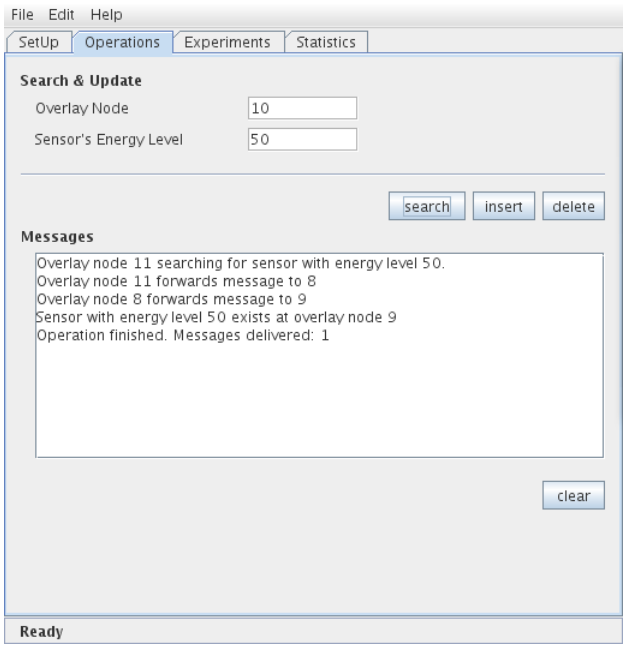


Figure 9: The tab "Operations"

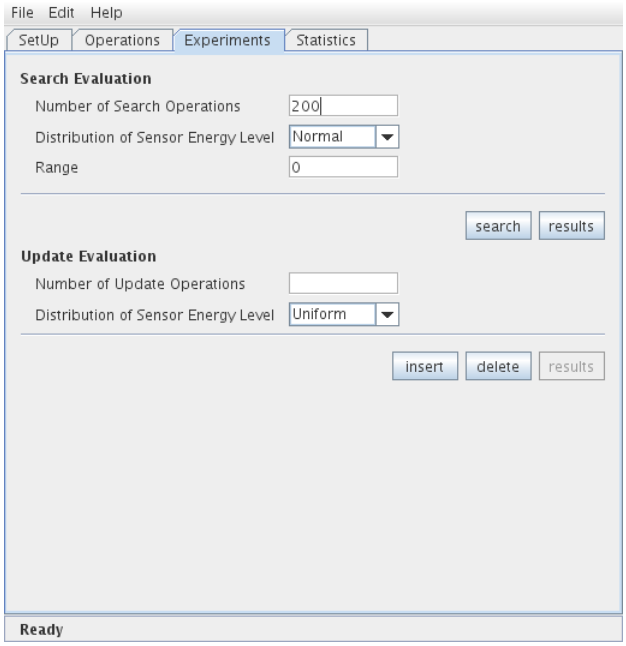


Figure 10: The tab "Experiments"

the system without the need of closing and reopening the simulator if we want to carry out several experiments with different number of peers and energy level distribution.

The second tab (see Figure 9) provides the ability to search, insert(join) / delete (leave) and update the energy level of a

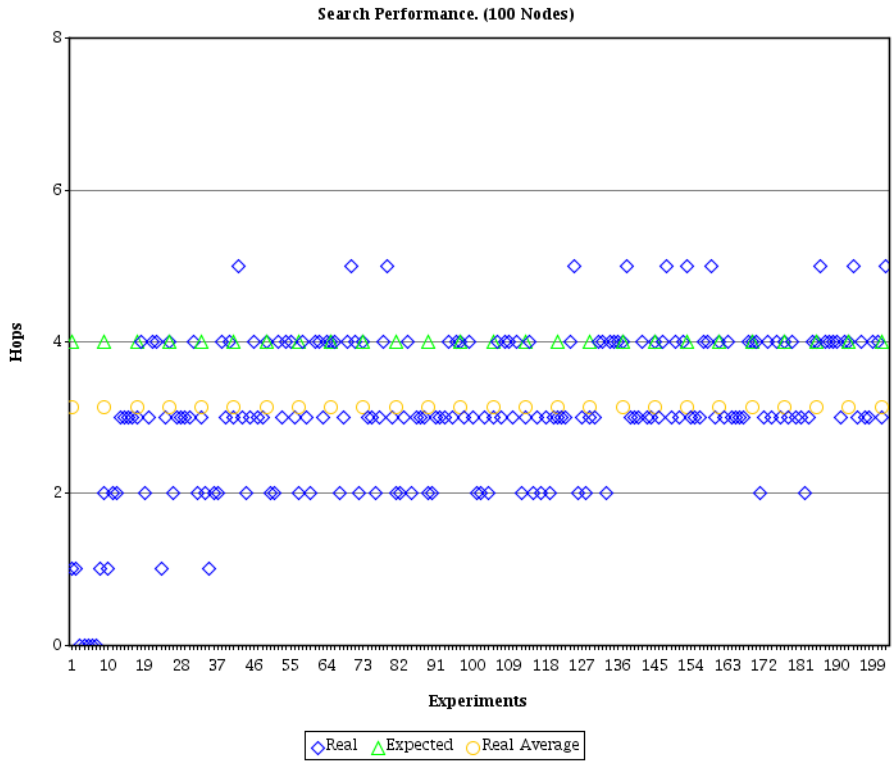


Figure 11: Lookup Performance Graph

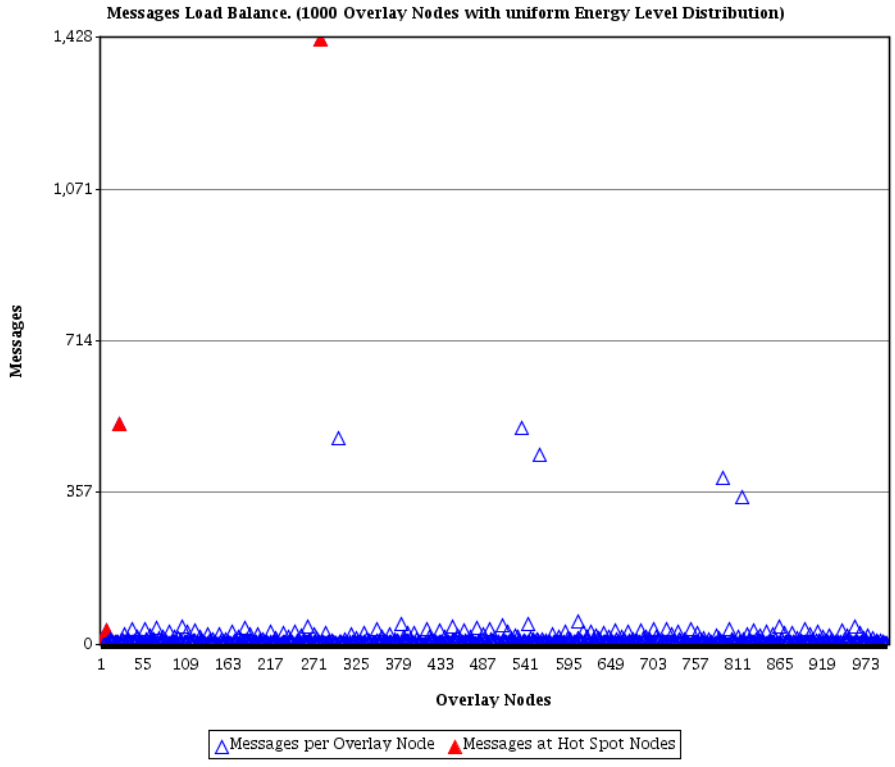


Figure 12: Load balance after 200 updates with uniform distribution.

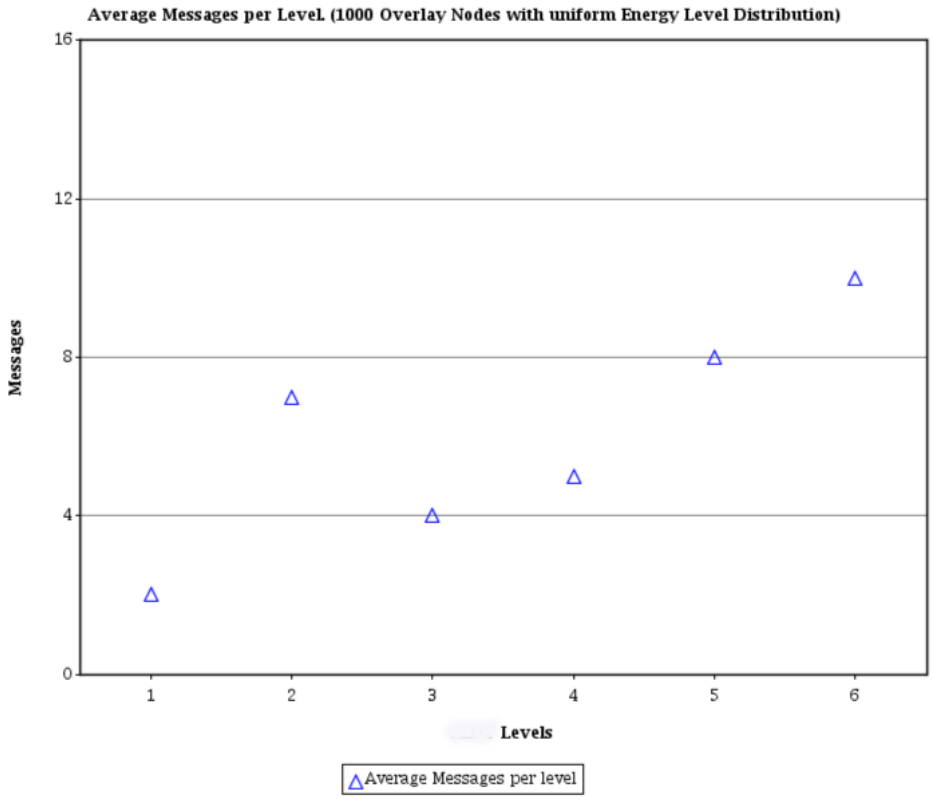


Figure 13: Average Messages per Level

sensor starting the procedure from any peer in the overlay. While one of these operations is being executed, appropriate messages are appearing at the bottom of this tab.

In the third tab (see Figure 10) the user can prosecute experiments to evaluate the efficiency of the lookup/update operations. There are two panels one for each operation where the user sets the number of the experiments and selects the distribution according to the energy-level keys of the sensors picked up for the experiments. After the termination of the experiments the user can see and save the chart that has been generated. In the forth tab - statistics - the user can see the current number of peers into the system, the number of sensors that have been stored over the peers and the range of sensors' energy level that we can store in the overlay. This tab represents also performance statistics such as the minimum, the maximum and the average path of the total operations that have been performed. Furthermore, this tab generates a chart with the load-balancing over the peers (see Figure 12), the number of messages that have been forwarded by each peer (see Figure 11)and the number of messages per tree level (see Figures 13).

In the most of cases, SART outperforms TChord by a wide margin. As depicted in Figures 14, 15 and 16 our method is almost 2 times faster for $b = 2$, 4 times faster for $b = 4$ and 5 times faster for $b = 16$. As a consequence we have a performance improvement from 50% to 80%. The results are analogous with respect to the cost of range queries as depicted in Figures 17, 18, 19, 20, 21 and 22.

In case $Query\_Range\_Length < Cluster\_Peer\_Key\_Range$ and $b = 2$, we have an 25% improvement, however, when $Query\_Range\_Length > Cluster\_Peer\_Key\_Range$, SART and TChord have almost similar performance behaviour.

In case $Query\_Range\_Length < Cluster\_Peer\_Key\_Range$ and $b = 4$, we have an 50% improvement, however, when $Query\_Range\_Length > Cluster\_Peer\_Key\_Range$ the improvement of our method downgrades to 13.15%.

In case $Query\_Range\_Length < Cluster\_Peer\_Key\_Range$ and $b = 16$, we have an 52.7% improvement, however, when $Query\_Range\_Length > Cluster\_Peer\_Key\_Range$ the improvement of our method downgrades to 21.05%.

Figures 23, 24 and 25 depict the cost of update queries. In particular, for $b = 2, 4, 16$, we have an improvement of 37.5%, 75% and 87.5% respectively.

Finally, Figure 26 depicts the cost of updating the routing tables, after peer join/leave operations. For *bad* or non-smooth distributions, like *powlow*, we have an 23.07% improvement. However, for more smooth distributions like *beta*, *normal* or $uniform$ the improvement of our method increases to 38.46%.

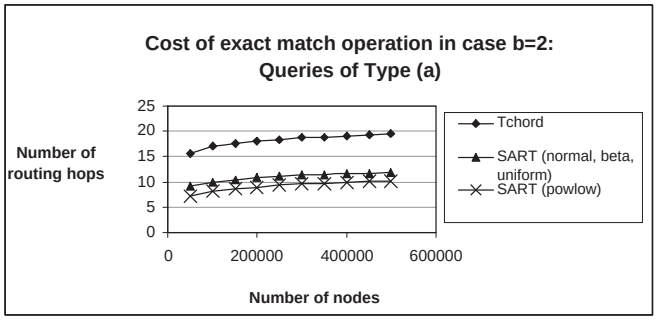


Figure 14: Cost of Exact Match Queries in Case b=2

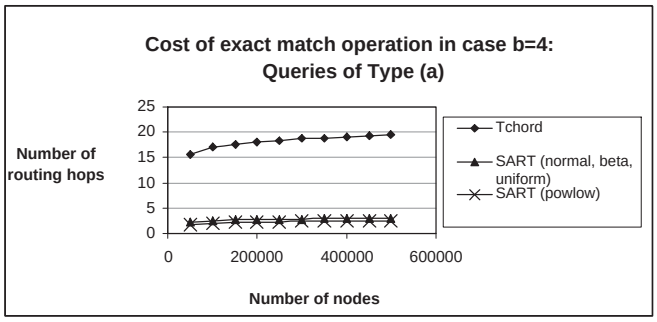


Figure 15: Cost of Exact Match Queries in Case b=4

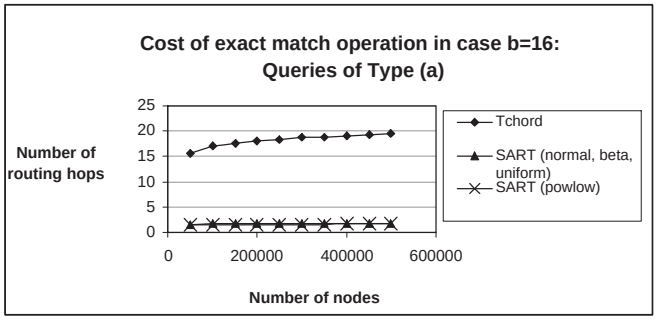


Figure 16: Cost of Exact Match Queries in Case b=16

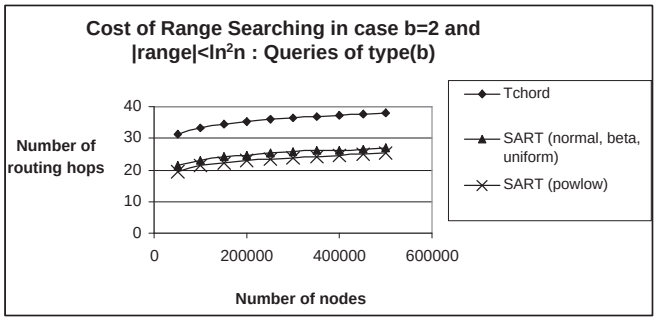


Figure 17: Cost of Range Queries in Case b=2 and $Query\_Range\_Length < Cluster\_Peer\_Key\_Range$

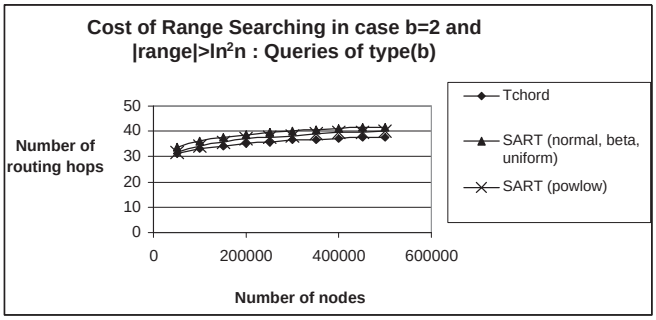


Figure 18: Cost of Range Queries in Case b=2 and $Query\_Range\_Length > Cluster\_Peer\_Key\_Range$

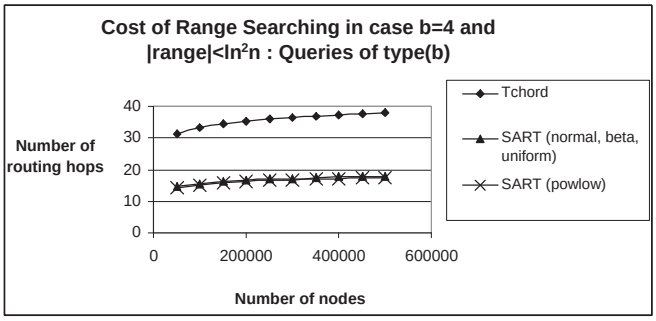


Figure 19: Cost of Range Queries in Case b=4 and $Query\_Range\_Length < Cluster\_Peer\_Key\_Range$

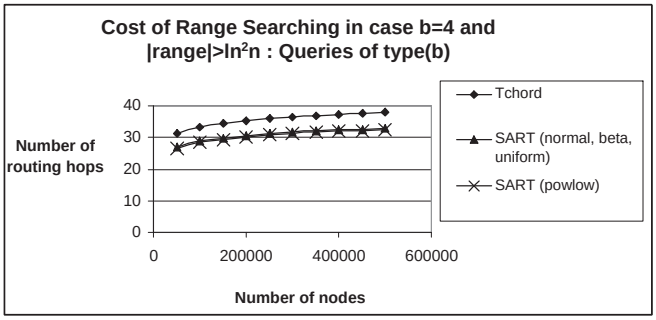


Figure 20: Cost of Range Queries in Case b=4 and $Query\_Range\_Length > Cluster\_Peer\_Key\_Range$

5. CONCLUSIONS

We considered the problem of constructing efficient P2P overlays for sensornets providing "Energy-Level Application and Services". On this purpose we designed SART, the best-known dynamic P2P overlay providing support for processing queries in a sensornet. We experimentally verified this performance via the D-P2P-Sim framework.

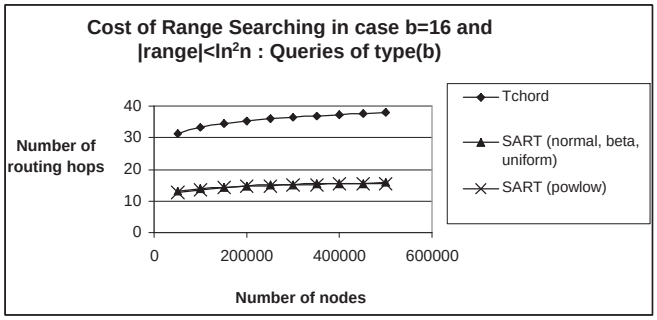


Figure 21: Cost of Range Queries in Case b=16 and $Query\_Range\_Length < Cluster\_Peer\_Key\_Range$

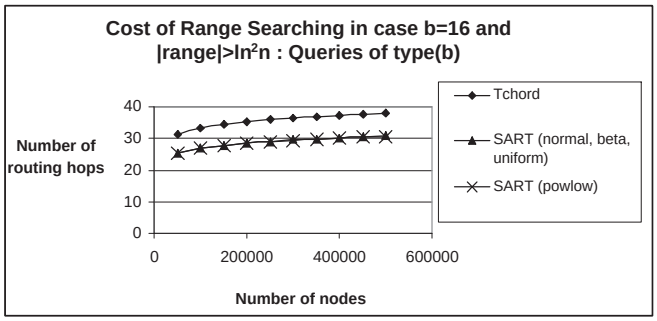


Figure 22: Cost of Range Queries in Case b=16 and $Query\_Range\_Length > Cluster\_Peer\_Key\_Range$

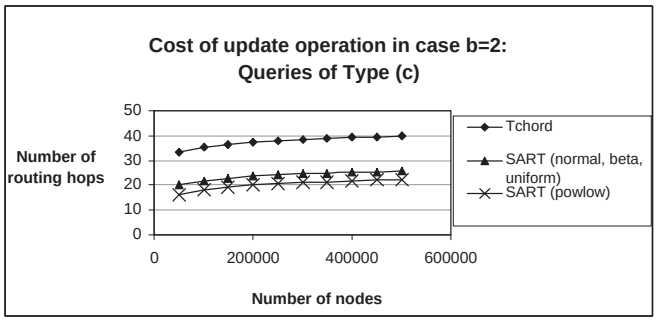


Figure 23: Cost of Update Queries in Case b=2

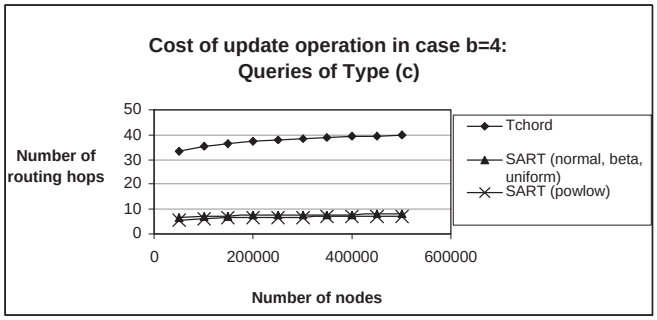


Figure 24: Cost of Update Queries in Case b=4

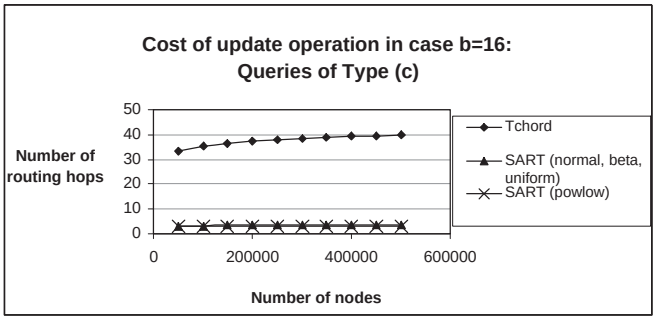


Figure 25: Cost of Update Queries in Case b=16

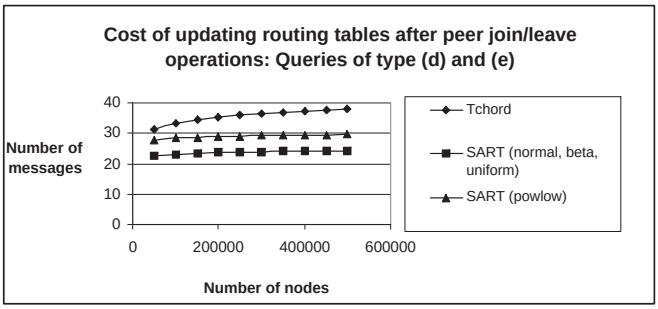


Figure 26: Cost of updating routing tables, after peer join/leave operations: The Cost is independed on parameter b

6. REFERENCES

[1] Muneeb Ali and Koen Langendoen, A Case for Peer-to-Peer Network Overlays in Sensor Networks, International Workshop on Wireless Sensor Network Architecture(WWSNA'07), pages 56-61, Cambridge, Massachusetts, USA, 2007.

[2] M. Ali and Z. A. Uzmi., CSN: A network protocol for serving dynamic queries in large-scale wireless sensor networks. In 2nd CNSR'04, pages 165-174, Fredericton, N.B, Canada, 2004.

[3] J. F. Buford, H. Yu, and E. K. Lua. P2P Networking and Applications. Morgan Kaufman Publications, California, 2008.

[4] M. Caesar, M. Castro, E. B. Nightingale, G. O'Shea, and A. Rowstron., Virtual Ring Routing: Network routing inspired by DHTs. In ACM SIGCOMM'06, pages 351-362, Pisa, Italy, 2006.

[5] Crainiceanu, A., Linga, P., Gehrke, J. and Shanmugasundaram, J., P-Tree: A P2P Index for Resource Discovery Applications, WWW'04, pages 390-391, New York, NY, USA, 2004.

[6] D. Clark, C. Partridge, R. T. Braden, B. Davie, S. Floyd, V. Jacobson, D. Katabi, G. Minshall, K. K. Ramakrishnan, T. Roscoe, I. Stoica, J. Wroclawski, and L. Zhang., Making the world (of communications) a different place. ACM SIGCOMM'05 CCR, 35(3):91-96, Philadelphia, PA, 2005.

[7] M.Demirbas, A.Arora, and M.Gouda., A pursuer-evader game for sensor networks. Sixth Symposium on Self- Stabilizing Systems(SSS'03), pages

1-16, San Francisco, CA, USA, 2003.

[8] Murat Demirbas, Hakan Ferhatosmanoglu, Peer-to-Peer Spatial Queries in Sensor Networks, IEEE Proceedings of the 3rd International Conference on Peer-to-Peer Computing, pp. 32-40, Linkoping, Sweden, 2003.

[9] M. Gerla, C. Lindemann, and A. Rowstron., P2P MANET's - new research issues. In Dagstuhl Seminar Proceedings, number 05152, Germany, 2005.

[10] H. V. Jagadish, B. C. Ooi, K. L. Tan, Q. H. Vu and R. Zhang., Speeding up Search in P2P Networks with a Multi-way Tree Structure, ACM SIGMOD'06, pages 1-12, Chicago, Illinois, 2006.

[11] H. V. Jagadish, B. C. Ooi, and Q. H. Vu., Baton: A balanced tree structure for peer-to-peer networks. In Proceedings of the 31st VLDB'05 Conference, pages 661-672, Trondheim, Norway, 2005.

[12] A. Kaporis, C. Makris, S. Sioutas, A. Tsakalidis, K. Tsichlas, and C. Zaroliagis. Improved Bounds for Finger Search on a RAM. *Algorithms*, Vol. 2832:325-336, 2003.

[13] A.Mainwaring, J.Polastre, R.Szewczyk, D.Culler, and J. Anderson. Wireless sensor networks for habitat monitoring. ACM Int. Workshop on Wireless Sensor Networks and Applications, September 2002.

[14] S.Sioutas, NBDT: An efficient p2p indexing scheme for web service discovery, Journal of Web Engineering and Technologies, Vol. 4 (1), pp 95-113, 2008.

[15] S. Sioutas, K. Oikonomou, G. Papaloukopoulos, M. Xenos, Y. Manolopoulos, "An Optimal Bipartite P2P Overlay for Energy-Level Queries in Sensor Networks", Proceedings of the ACM international Conference on Management of Emergent Digital Ecosystems - ACM Special Interest Group on Applied Computing (ACM-SIGAPP MEDES 2009), Lyon, France, pp.361-368.

[16] S.Sioutas, G. Papaloukopoulos, E. Sakkopoulos, K. Tsichlas, Y. Manolopoulos and P. Triantafillou "Brief Announcement: ART:Sub-Logarithmic Decentralized Range Query Processing with Probabilistic Guarantees", In Proceedings of Twenty-Ninth Annual ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing (ACM PODC 2010), Zurich, Switzerland July 25-28, pp. 118-120, 2010.

ABOUT THE AUTHORS:

Spyros Sioutas was born in Greece, in 1975. He graduated from the Department of Computer Engineering and Informatics, School of Engineering, University of Patras, in December 1997. He received his Ph.D. degree from the Department of Computer Engineering and Informatics, in 2002. He is now an Assistant Professor in Informatics Department of Ionian University. His research interests include Data Structures and Databases, P2P Data Management, Data Warehouses and Data Mining, Computational Geometry, GIS and Advanced Information Systems. He has published over 100 papers in various scientific journals and refereed conferences.

Alexandros Panaretos is currently a PhD Student at the Department of Informatics, Ionian University. He obtained his BEng in Software Engineering from the Computer Science Department, University of Wales Aberystwyth in 2001 and his MSc in E-Commerce Technology from the Computer Science Department , University of Essex in 2002. His research interests focuses on P2P Data Management, GIS Systems and Social Networks.

Ioannis Karydis was born in Athens, Greece in 1979. He received a BEng (2000) in Engineering Science & Technology from Brunel University, UK, an MSc (2001) in Advanced Methods in Computer Science from Queen Mary University, UK and a PhD (2006) in Mining and Retrieval Methods for Acoustic and Symbolic Music Data from the Aristotle University of Thessaloniki, Greece. He has contributed to more than 35 academic publications and currently is a contract lecturer at the Ionian University, Greece. His research interests include Networking Data Management, Music Databases, Music Information Retrieval (indexing & searching), Music Genre Classification, Musical Similarity using Contextual Information, Continuous Querying in musical streams, Cultural Information Systems and Privacy Issues in Databases.

Dimitrios Tsoumakos is an Assistant Professor in the Department of Informatics of the Ionian University. He is also a senior researcher at the Computing Systems Laboratory of the National Technical University of Athens (NTUA). He received his Diploma in Electrical and Computer Engineering from NTUA in 1999, joined the graduate program in Computer Sciences at the University of Maryland in 2000, where he received his M.Sc. (2002) and Ph.D. (2006).
His research interests lie in the area of distributed data management, particularly in designing and implementing adaptive and scalable schemes for big data storage and indexing. He is also involved in Database research, especially in designing distributed indexing schemes for sharded databases. His most recent projects relate to automatic elasticity provisioning for NoSQL engines and scalable RDF query processing using NoSQL and MapReduce.

Giannis Tzimas is currently an Assistant Professor in the Department of Applied Informatics in Management & Economy of the Technological Educational Institute of Mesolonghi. Since 1995, he is also an adjoint researcher in the Graphics, Multimedia and GIS Lab, Department of Computer Engineering & Informatics of the University of Patras. He graduated from the Computer Engineering and Informatics Department in 1995 and has participated in the management and development of many Research and Development projects funded by national and EU resources, as well as the private sector. His research activity lies in the areas of Computer Networks, Web Engineering, Web Modelling and Bioinformatics. He has published a considerable number of articles in prestigious national and international conferences and journals.

Dimitrios Tsolis is a lecturer of the Cultural Heritage Management and New Technologies Department of the University of Western Greece. He is responsible for the courses for Introduction to Informatics and Networks, Internet and Semantic Web Technologies and Human Computer Interaction. His Ph.D. was focusing on Software Development and Engineering for Advanced Information Systems and Networks especially focusing on Digital Rights Management. He has over 70 publications in scientific and international Journals, Conferences and Technical Reports. He has also supervised or participated to more than 30 R&D Projects in the area of Computer Science.