Available online at www.sciencedirect.com
SCIENCE DIRECT®
Computers and Chemical Engineering 28 (2004) 459–478
Computers & Chemical Engineering
www.elsevier.com/locate/compchemeng

Multi-objective process design in multi-purpose batch plants using a Tabu Search optimization algorithm

L. Cavin [a], U. Fischer [a,*], F. Glover [b], K. Hungerbühler [a]

[a] *Department of Chemical Engineering, ETHZ, Zurich 8093, Switzerland*
[b] *Leeds School of Business, University of Colorado, Boulder, Colorado 80309, USA*

Received 26 November 2002; received in revised form 31 July 2003; accepted 31 July 2003

Abstract

Chemical batch processes are typically used for the production of speciality chemicals and pharmaceuticals. Due to the still growing importance of this type of processing, design methods are required that take into account the special requirements and constraints in the corresponding production facilities. We developed a method that optimizes the design of a single chemical process to be implemented in an existing multi-purpose batch plant, in which a well-defined set of equipment units is available for realizing this process. In the optimization, three objectives with different priorities are considered. A flexible metaheuristic algorithm, Tabu Search (TS), has been implemented to solve this multi-objective combinatorial non-linear problem. We started from a basic form of TS to determine the effectiveness of this version as well as establish the relative strengths and weaknesses of first level TS strategies. Our investigation includes a thorough examination of algorithm parameters and of implementation issues to identify algorithm settings that can handle the whole class of problems considered. Overall, we concluded that the basic form of TS—using fixed default settings—exhibits highly attractive performance features for solving the problems at hand. Moreover, comparison with a multi-start steepest descent algorithm shows that a basic TS approach conducts a global search more effectively. As illustrated by three case studies, the new method is well suited for identifying optimal designs of a chemical process to be implemented in an existing multi-purpose batch plant. The approach is particularly suited for considering multiple prioritized objectives and for enabling the use of external (e.g. commercial) batch process simulation software as a black-box model for the process evaluations.

Keywords: Batch process design; Multi-objective combinatorial optimization; Tabu Search; Multi-purpose plant

1. Introduction

1.1. Batch process design

Speciality chemicals and pharmaceutical products are typically produced in batch processes. Corresponding plants are often classified as multiproduct batch plants, in which every product follows the same sequence through all the process steps, or as multi-purpose batch plants, in which each product follows its own distinct processing sequence by using the available equipment in a product-specific layout (Rippin, 1983). In practice combinations of these two limiting scenarios might also arise. Multi-purpose plants can be used in two main modes: either only one production runs in the plant at a given time or many processes run concurrently. Some multi-purpose plants consist of discrete but flexible production lines that are independent from each other.

Because of the escalating importance of these types of chemical processes, in recent years increased research efforts have been undertaken to develop design methods for batch processes. Many methods deal with the grassroot design of multiproduct or multi-purpose batch plants and include the equipment sizing problem (Grossmann & Sargent, 1979; Papageorgaki & Reklaitis, 1990; Sparrow, Forder, & Rippin, 1975; Suhami & Mah, 1982; Voudouris & Grossmann, 1992). In most cases, the authors only consider the case where many productions run concurrently.

Relatively few publications have been presented that deal with the optimum design of a single batch process. For grassroot design, Loonkar and Robinson (1970) described a

* Corresponding author. Tel.: +41-1-6327142; fax: +41-1-6321189.
E-mail addresses: cavin@tech.chem.ethz.ch (L. Cavin), ufischer@tech.chem.ethz.ch (U. Fischer), fred.glover@colorado.edu (F. Glover), hungerb@tech.chem.ethz.ch (K. Hungerbühler).

0098-1354/$ – see front matter © 2003 Elsevier Ltd. All rights reserved.
doi:10.1016/j.compchemeng.2003.07.002

Nomenclature

B	block matrix containing sequences of tasks to be conducted in the same equipment unit(s)
B_R	block-to-recipe one-to-many relationship matrix
C	connectivity constraints matrix
E	equipment matrix
K	recipe indexes vector (list of tasks belonging to one block)
L	allocated equipment units matrix, L is a solution to the design problem
O	operation library matrix, reference containing all possible task classes
P	standard operating ranges library matrix
R	recipe matrix consisting of a list of tasks to be conducted
S	eligible equipment units vector
T	tabu tenure
U	eligible equipment classes vector
X	secondary recipe matrix, subset matrix of R for the tasks indicated in K
y	temporary binary variable indicating if the considered task is a transfer (1) or not (0)
Z	sequence matrix allowing to handle branching recipes

Subscripts

i	main counter across the block matrix B
j	secondary counter across the block matrix B

procedure for the cost optimum design and apparatus sizing of a single batch process, while Takamatsu, Hashimoto, and Hasebe (1982) presented a similar approach that considers the possibility of intermediate storage. Yeh and Reklaitis (1987) presented a method for the preliminary grassroot design of a single batch process including an approximate sizing procedure. Mauderli and Rippin (1979) developed a method for planning and scheduling in multi-purpose batch plants. While they consider many concurrent productions, their first step consists of the generation of design alternatives for the production of single products. To obtain the different design alternatives they take account of the plant specifications (number and size of available equipment units) and the process requirements (which equipment units can be used for the different process tasks). These alternative designs are not optimized but generated using a heuristic procedure that allows the selection of promising designs while ignoring designs with a low performance. The objective of these researches is to optimize n subsequent batches that can follow different paths in the plant. However, for safety, regulatory and controlling reasons, it is often preferred to have all batches using the same path. Under this perspective, the objective changes from the *optimum schedule of n subsequent batches* to the *design of the single most efficient batch*. To our knowledge no method to date has focused on this particular design problem.

In order to identify an optimal solution for this problem it is important to consider in the design procedure all details and existing constraints such as equipment specifications (e.g. range of operating temperature and pressure, lining material, special supply pipes, the floor at which each equipment is located), design constraints (e.g. feasible and infeasible connection of equipment units), and process requirements (e.g. reaction mixture that cannot be safely transferred, thus forcing several operations to be conducted in the same equipment unit). This is the approach taken in the method presented here. Based on pre-defined rules (representing heuristics) and options selected by the user, specified operating instructions for a chemical process to be designed are automatically analyzed with regard to design requirements and constraints: suitable equipment units are assigned to each operation, feasible and infeasible transfers are identified, and operation blocks are determined that will be conducted in the same equipment unit. This results in a superstructure for which the optimum design can be identified.

1.2. Integer non-linear optimization

The problem that has to be solved is one of combinatorial optimization, a pure integer problem: the optimization involves decision variables that take only integer values, without any continuous variables. Even when the user selects to use short-cut models to adapt operation durations, non-linear (e.g. stepwise) functions are evaluated to compute the objective function, rendering this problem a non-linear integer system. Such systems are known to be NP-Complete and cannot be solved in polynomial time. Various algorithms and methods have been developed to tackle similar problems that can be classified in three main categories: heuristics, mathematical programming and metaheuristic algorithms.

Heuristics (of the classical kind) do not actually solve the optimization problem, but aim at finding “good” solutions by following a set of rules. In the chemical process design field, Douglas (1985) has developed a method for hierarchical process synthesis that relies on sets of rules at different stages during process development (see also Siirola, 1996). The computer-oriented implementation of such systems usually takes the form of an Expert System, as for example presented by Kirkwood, Locke, and Douglas (1988). Such methods are good in finding quickly and reliably a good solution that can be used for example as starting point for more advanced metaheuristic algorithms.

The mathematical programming methods (sometimes called “exact methods”) are rigorous optimization techniques aimed at solving the Mixed Integer Non-Linear Problem (MINLP) formulation of the design problem (Grossmann, 1985). These techniques use usually algorithms derived from Branch and Bound or Outer Approximation, as discussed by Grossmann and Kravanja (1995) and Skrifvars, Harjunkoski, Westerlund, Kravanja, and Porn

(1996). They have been used extensively in process design (see e.g. Ciric & Floudas, 1990; Hostrup, Gani, Kravanja, Sorsak, & Grossmann, 2001; Papalexandri & Pistikopoulos, 1996). A recent review of these methods has been published by Grossmann, Caballero, and Yeomans (1999) and their applicability, limitations and potentials are discussed by Grossmann and Daichendt (1996), Gruhn and Noronha (1998), and Kallrath (2000).

The last category, the metaheuristic algorithms, are based on one (or several) initial solution(s) and a progressive (though not necessarily uniform) improvement of their quality. A popular example of a metaheuristic is the genetic algorithm (GA) approach. This scheme is characteristically based on a population of solutions that are combined (using a crossover operation) and randomly modified (using a mutation operation) according to their fitness (objective function value), leading to a "natural" selection and a discrimination in favor of solutions with a good fitness. In the process design field, Fraga and Matias (1996) for example used a genetic algorithm to optimize the design of a distillation system.

Another metaheuristic method is simulated annealing (SA) developed by Kirkpatrick, Gelatt, and Vecchi (1983). SA randomly modifies an initial solution, and always accepts downhill (improving) moves when encountered. Other moves are accepted if they satisfy a condition that depends on the advancement of the algorithm (the Metropolis criterion, expressed as "temperature"), which endows the method with an ability to accept moves that can escape from a local optimum. In the process design field, SA has been used to design separation systems by Floquet, Pibouleau, and Domenech (1994) and to handle overall process design by Chaudhuri and Diwekar (1996).

A third metaheuristic method is the Tabu Search (TS), developed by Glover (1977). TS makes use of adaptive memory to escape local minima. TS has had numerous successful applications in recent years (for example, the website http://www.tabusearch.net lists over a thousand presentations and articles on the method), but to our knowledge it has been used only once in the field of batch process design by Wang, Quan, and Xu (1999) for the problem of the grassroot design of multiproduct batch processes.

Within the present project the goal was to enable the use of external batch simulation programs (black-box optimization). Black-box models are exceedingly difficult to handle in conjunction with mathematical programming approaches, therefore making it attractive to employ a metaheuristic algorithm. Gross and Roosen (1998) have tackled a similar problem (continuous process design with a black-box external simulation package) and have chosen a genetic algorithm. However, GA approaches have encountered significant difficulties when confronted with problems that contain complex constraints, which are a predominant feature in the problems we face. The limitation of GAs in these settings arises from the inability to implement crossover operations that generate valid designs. Recourse to penalty approaches and ad hoc "repair operators" as an attempted remedy entails a risk of spending most of the computational effort in handling invalid solutions, making GAs unsuitable for our present application. Another evolutionary approach, path relinking, offers a greater capability for handling constraints. This approach is often coupled with Tabu Search (and in fact, emerged from the same origins as Tabu Search), thereby motivating us to look at an implementation of TS in this study. Additional reasons for choosing Tabu Search, and for choosing a multi-start descent procedure to compare it against (in contrast to simulated annealing, for example), are indicated in the next section.

1.3. Objectives of this research

Since very limited experience is available with regard to using Tabu Search in chemical process design, we decided to conduct a "grass roots" investigation of the algorithm to determine its performance characteristics in this setting.

The objectives of the research presented here were to:

- Investigate the suitability of TS for the chemical process design problem discussed above, and to determine parameters and settings that have the greatest relevance for a basic TS implementation.
- Establish the relative strengths and weaknesses of first level TS strategies for solving this class of problem.
- Examine the suitability of TS to handle multiple prioritized objectives.
- Determine default parameter settings that can handle the whole class of problems with the highest probability of finding the global optimum while using a modest amount of computational effort.

Our major aim being to study the suitability of TS for solving a new class of problems, we concentrated on examining a basic form of Tabu Search exclusive of deeper TS strategies that comprise an integral part of more advanced versions. Additionally, we want to establish how a simple TS implementation compares with a principal alternative approach (multi-start descent) that has been documented to have good performance characteristics and strong convergence properties. As previously noted, we have not undertaken to create a computational comparison of our TS approach with simulated annealing. The reason for instead choosing multi-start descent (sometimes called "iterated descent" or "generalized hill climbing") is the demonstration by recent studies that this approach may dominate simulated annealing both theoretically and empirically (see e.g. Jacobson, 2002; Lourenco, Martin, & Stutzle, 2002).

Finally, we intend to provide insight into the choice of higher level strategies that are likely to produce the greatest improvements.

Against this backdrop, we investigate the effectiveness of our Tabu Search approach by using three real world case studies.

2. Batch process design method

The problem covered in this paper can be defined as follows:

- *Given*:
  - (a) Recipe
    - a. Recipe of the product expressed as a series of chemical/physical tasks.
    - b. Capacity requirements for each task per unit of final product.
    - c. Base duration of each task at the input scale.
    - d. Recipe constraints (specific rules about how tasks can be combined).
  - (b) Equipment
    - a. Available equipment units and their detailed specifications.
    - b. Connectivity constraints (governing links between equipment).
  - (c) General heuristics
    - a. Equipment classes suited for processing each task class.
    - b. Design heuristics (built-in and user options).
    - c. Relationships between task processing time and the batch size.
  - (d) One or more objective functions
- *Determine*:
  - An optimal layout for the process—allocating equipment units to tasks and decide on the design options (in parallel, in series).

A general overview of the method is presented in Fig. 1. The routines for recipe analysis, superstructure generation, process simulation, and optimization have been implemented in a MATLAB® program. The process simulations can also be conducted in an external (e.g. commercial) batch process simulator. The different input and methodological steps are summarized below using a relational data structure formulation, the tuple calculus (Date, 1995). Table 1 provides a list of the symbols used in the mathematical formulation.

The first input is the base recipe *R*. The recipe is given as a table (which is converted to a matrix in MATLAB®) containing vertically the physico-chemical tasks to be conducted; the vertical position, also called the *index*, represents the position of the given task in the task sequence. Branched recipes are handled by including an additional matrix *Z* (not included in the equations below) to describe the sequence of the rows in the recipe (Eq. (11) (see below) would be modified to take matrix *Z* into consideration).

Each row of *R* first indicates the nature of the task (*R.OperationClassID*), the base volume (*R.Volume*) and the estimated base duration (*R.Duration*) required for the task. All column names ending in *ID* are relations to lists of available options stored in library matrices. For instance, *R.OperationClassID* refers to the operation class matrix *O* containing all supported types of tasks (e.g. reaction, distillation ...). Operating temperature (*R.Temperature*), pressure (*R.Pressure*) and required lining materials (*R.LiningID*) are then given, and the matrix is completed by flags (*R.Flag*) indicating constraints on possible designs, duration adaptation rules and algorithm options.

Based on *R.Flag*, the recipe is condensed in a block matrix *B* (step "Recipe Analysis" in Fig. 1). Each row of *B* (i.e. each operation block) contains a list of tasks that *must* be conducted in the same equipment unit(s). The matrix stores the largest volume attained during the block, as well as the highest pressure and temperature reached.

According to the duration adaptation rules in *R.Flag*, *B* also contains two duration components: *B.ConstantDuration* contains the volume independent part of the aggregated duration of the block, while *B.LinearDuration* contains the volume dependant part. If the time adaptation rules are neither linear nor constant for one task, as identified by *R.Flag*, this indication is propagated to *B.Flag*. Finally, the flags and the lining material requirements are summarized in each row.

A block-to-recipe one-to-many relationship matrix B_R is also set up in order to keep track of which tasks are contained in which block.

An illustrative example of the recipe matrices is given in Fig. 2. The matrix *R* is a transposition of the text recipe given and also includes heuristics which are summarized as flags. For instance, due to the presence of the flag "Do not transfer after operation" (flag 2) in the *Charge* operation, the two first operations are grouped in the block matrix *B*. The flag "Cannot be conducted in series" (flag 8) is not propagated to the whole group, as the group may be conducted in series as long as the transfer happens during the reaction. On the other hand, the flags "Cannot be conducted in parallel" (flag 4), or "Special Scale-up rules" (flag 32—in use, e.g. for multi-drop centrifuges) would for instance have been propagated. The operation indexes given in the B_R matrix are the row numbers of the operations in matrix *R*.

The equipment list is similarly represented by a matrix *E*. Each row of *E* represents a single equipment unit, defined

Table 1
Tuple calculus symbols and operators used in the mathematical formulation

Symbol	Meaning
$a.b$	Column b of matrix a
$(a\|a.c = 1)$	Take rows of a where column $c = 1$
$(a.b\|a.c = 1)$	Take only column b (same rows as above)
$\in$	Is contained in (single element)
$\supset$	Contains (multiple elements)
$\cup$	Union
$\cap$	Intersection
$\leftarrow$	Assign a subset of a matrix to itself (filtering)
$\varnothing$	Empty (no elements)

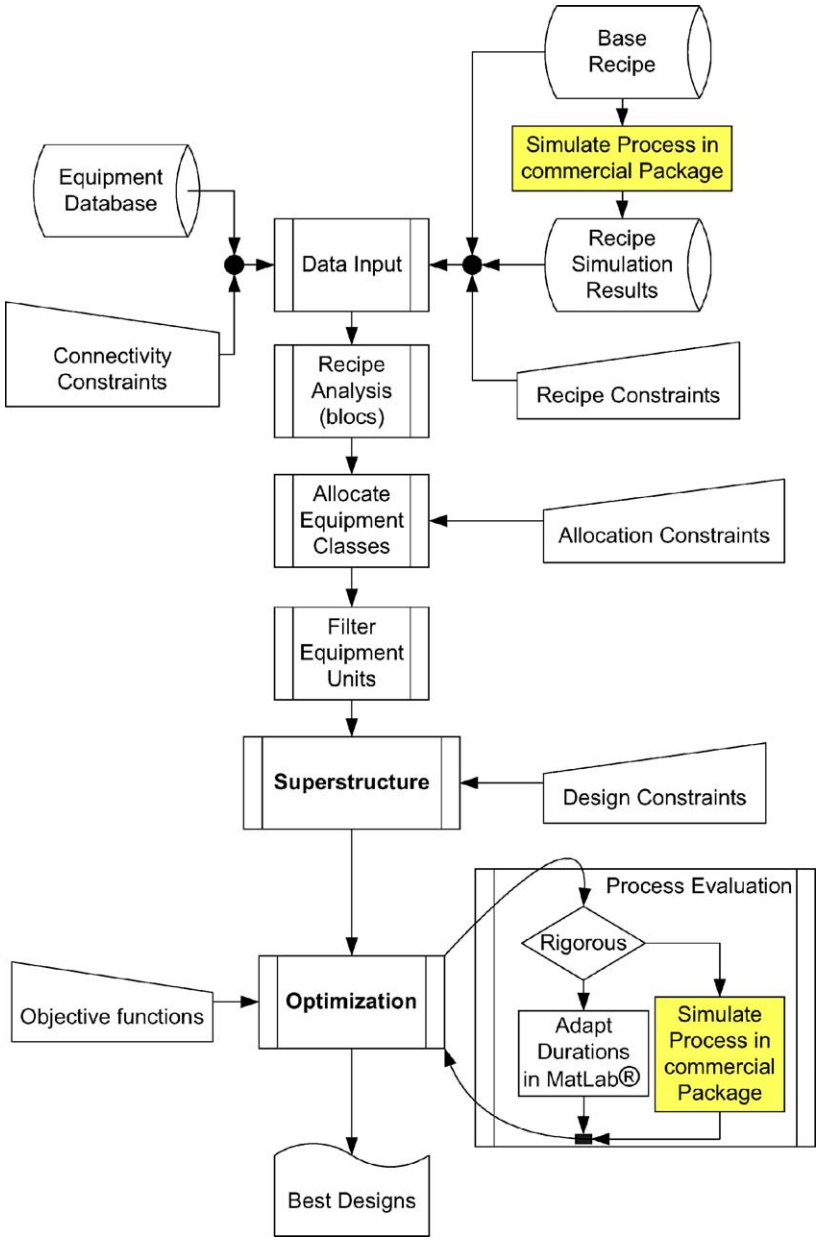

Fig. 1. Overview of the method; gray boxes indicate optional usage of external software.

by a unique number *E.EquipmentID*. If multiple equipment units belong together (for example, a condenser on the top of a reactor), an internal link to the master *EquipmentID* (e.g. the reactor) is given in the slave equipment (e.g. the condenser) record. Each equipment unit is defined by a class defining the type of equipment (e.g. reactor, centrifuge, condenser ...), the volume, the lining material (*E.LiningID*, pointing to the same library matrix as *R.LiningID*) the operating range being indicated by a pointer *E.TP_rangeID* to the standard operating ranges matrix *P* that contains maxima and minima for both pressure and temperature. Finally, the physical location in the plant is defined by the floor.

Feasible equipment interconnections are summarized in the connectivity matrix *C* that contains many-to-many relationships based on the *EquipmentID*. The equipment class allocation to each block of tasks (third step in Fig. 1) is based on the allocation matrix *A*. It contains many-to-many relationships indicating which task class(es) can be conducted in which equipment class(es). The equipment class allocation proceeds according to Eqs. (1)–(4) (the subscript i is the main counter across the block matrix *B*).

$$\textit{recipe indexes} := K_i = (B_{\mathrm{R}}.\textit{RecipeIndex} | B_{\mathrm{R}}.\textit{BlockID} = i) \tag{1}$$

$$\textit{recipe operations} := X_i = (R | R.\textit{index} \in K_i) \tag{2}$$

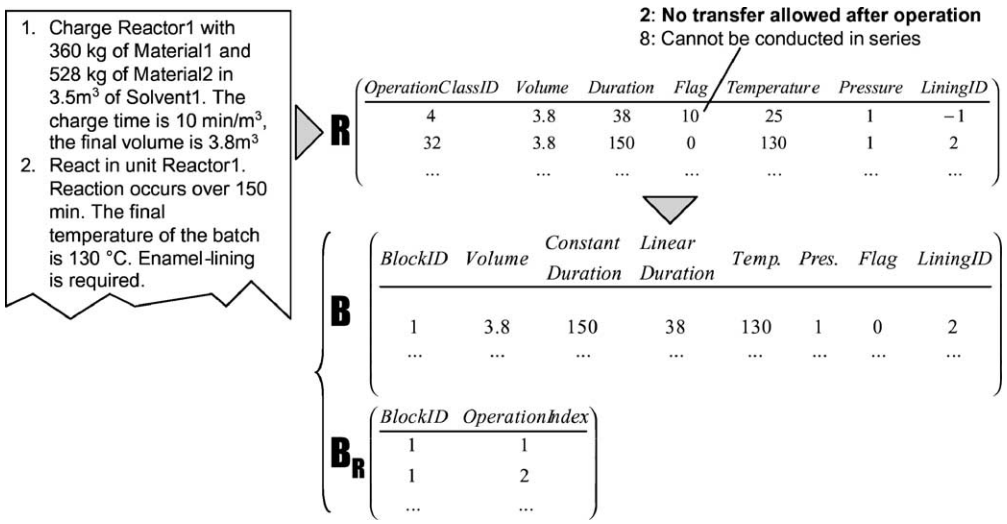

Fig. 2. Illustrative example of the recipe matrix R and the resulting block matrices B and B_R, based on a fragment of a text recipe (see text for further explanations).

equipment classes :=

$$U_i = \left(A.EquipmentClassID | A.OperationClassID \supset \bigcup X_i.OperationClassID\right) \quad (3)$$

eligible equipment units :=

$$S_i = (E | E.EquipmentClassID \in U_i) \quad (4)$$

In the next step, the eligible equipment units S_i are filtered according to the lining material and operating conditions requirements, as shown in Eqs. (5)–(9).

$$S_i \leftarrow (S_i | (P.P\min | P.TPrangeID = S_i.TP\_range) \leq \min(X_i.Pressure)) \quad (5)$$

$$S_i \leftarrow (S_i | (P.P\max | P.TPrangeID = S_i.TP\_range) \geq \max(X_i.Pressure)) \quad (6)$$

$$S_i \leftarrow (S_i | (P.T\min | P.TPrangeID = S_i.TP\_range) \leq \min(X_i.Temperature)) \quad (7)$$

$$S_i \leftarrow (S_i | (P.T\max | P.TPrangeID = S_i.TP\_range) \geq \max(X_i.Temperature)) \quad (8)$$

$$S_i \leftarrow \left(S_i | S_i.LiningID \in \bigcap X_i.LiningID\right) \quad (9)$$

The superstructure is defined by combining these eligible equipment units for each task block with design rules and constraints. The first constraint is that no equipment unit can be used twice for the same batch, as shown in Eqs. (10) and (11) (the subscript j is a secondary counter on the block matrix B).

Allocated equipment units := L, $L_0 = \emptyset$,

$$L_{i=\text{transfer}} = \emptyset \quad (10)$$

$$S_i \leftarrow S_i - \sum_{j=1\rightarrow i-1} (L_j) + L_{i-1} \quad (11)$$

The connectivity constraints given in matrix C are taken into consideration in Eqs. (12) and (13) (y is a temporary binary variable indicating a transfer (1) or not (0)).

$$y = (X_{i-1} | X_{i-1}.Class \in Transfer\_Classes) \quad (12)$$

$$S_i \leftarrow \begin{cases} (S_i | S_i.EquipmentID \in (C.ToEquipmentID | C.FromEquipmentID \in L_{i-2})) \\ \quad \text{if } y \neq \emptyset \\ (S_i | S_i.EquipmentID \in (C.ToEquipmentID | C.FromEquipmentID \in L_{i-1})) \\ \quad \text{if } y = \emptyset \end{cases} \quad (13)$$

Finally, the remaining recipe constraints given in ($R.Flag$), that indicate which design can be used to implement each task, are taken into consideration in addition to those design constraints given by the user, as shown in Eqs. (14) and (15). (An example of such a design constraint is to indicate whether the reaction mixture can be properly separated into two equal volumes with the same concentration of all components, allowing the task to be conducted in parallel on multiple equipment units. A user design constraint might further indicate, for instance, that at most two equipment units can be used in parallel.)

$$L_i = one\ single\ element\ of\ S_i \quad (14)$$

$$L_i = \text{one of} \begin{cases} L_i + \text{one additional element of } S_i \\ \quad \text{only available if } 4 \notin B_i.Flag; \\ \quad L_i.DesignType = parallel \\ L_i + \text{one or several additional elements of } S_i \\ \quad \text{only available if } 8 \notin B_i.Flag; \\ \quad L_i.DesignType = series \\ L_i \\ \quad \text{always available}; \\ \quad L_i.DesignType = single \end{cases} \tag{15}$$

In Eqs. (14) and (15), choices are left open in formulations like "one single element of". All possible decisions for making such choices determine a valid process. As represented by the last step in Fig. 1, these decisions must be optimized in order to obtain the optimum design. This optimization is investigated in detail in the following sections.

The optimization involves three objective functions:

- Maximize the throughput, i.e. the batch size divided by the batch cycle time.
- Optimize the quality of the design, as evaluated by two indicators:
  - minimize the number of equipment units used, to discriminate in favor of simple designs;
  - minimize the number of floors the reaction mixture has to be pumped up, to discriminate in favor of top–down designs.

These objective functions are prioritized: a larger throughput is always preferred to a simpler design. Similarly, a simpler design is always preferred to one that is more "top–down." This means that the secondary objective function will only be evaluated when an equal throughput is produced by two designs, and similarly the third objective function is only evaluated if two designs produce the same throughput *and* use the same number of equipment units.

The primary objective function evaluation requires a process simulation of the recipe. This evaluation can be conducted in an external batch process simulation program. Alternatively, it can be conducted in MATLAB® using the constant duration (e.g. duration of a reaction is volume independent) or a linear time adaptation according to the volume, for instance for transfers or distillations. (Additionally, some specific rules called by *B.Flag* have been implemented for special adaptations of durations, like in a multiple-drop centrifuge where first the number of drops needed is calculated and then multiplied by the drop duration.)

3. Tabu Search

3.1. Introduction

Tabu Search is a metaheuristic method using a set of coordinated strategies for introducing and exploiting adaptive memory, in order to generate a sequence of solutions that contains a subsequence of progressively improving "best solutions found." Repetitively, modifications of the current solution are examined, and the one resulting in the best solution is chosen for the next iteration, even if this successor is worse than the present solution. The special memory processes continue to drive the method forward to discover regions that harbor one or more solutions better than the current best, if such solutions exist. Some forms of TS have a proof of finite convergence to optimality (Glover & Hanafi, 2002), but the most effective forms generally do not. A comprehensive description of TS with examples of applications is given in Glover and Laguna (1997).

The fundamental version of Tabu Search used in this study is depicted in the schematic flowchart of Fig. 3. In the following, the different rules and options for the algorithm will be discussed.

3.1.1. Initial solution

First an initial solution has to be provided. Usually there are advantages to starting from an initial solution that is of high quality, such as one proposed by experts or generated by preliminary heuristics.

A method providing multiple initial solutions can however be preferable: the constraints may make the solution space non-convex, and hence a "good" initial solution may be computationally very far (i.e. numerous moves) from the optimum. An efficient way of exploring the whole solution space is to restart with different initial solutions. They can be strategically generated to be diverse, or be generated by a randomization component. For instance, a strategy used by Wang et al. (1999) applies a random selection of the initial solution that is biased to cover previously unvisited regions of the optimization space as a means of providing a targeted diversification of the search (other more sophisticated diversification strategies are discussed by Glover and Laguna (1997)).

3.1.2. Move definition—neighborhood generation

The definition of the moves, i.e. the definition of the modifications of the current solution that can be done at each iteration, is highly problem-specific. The current solution combined with the moves defines a neighborhood of possible "next solutions." In general terms, larger neighborhoods afford an opportunity to encounter shorter paths (fewer moves) to reach an optimal solution. However, large neighborhoods require special candidate list strategies to isolate a subset of the neighborhood to be tested (see below), and unless the candidate lists are chosen effectively, the path actually selected may be quite long, requiring a large number of steps to reach an optimum (or perhaps never reaching an optimum at all). This is the first trade-off addressed in the parameterization of our TS approach. As explained in the following, such trade-offs also arise for other components of TS.

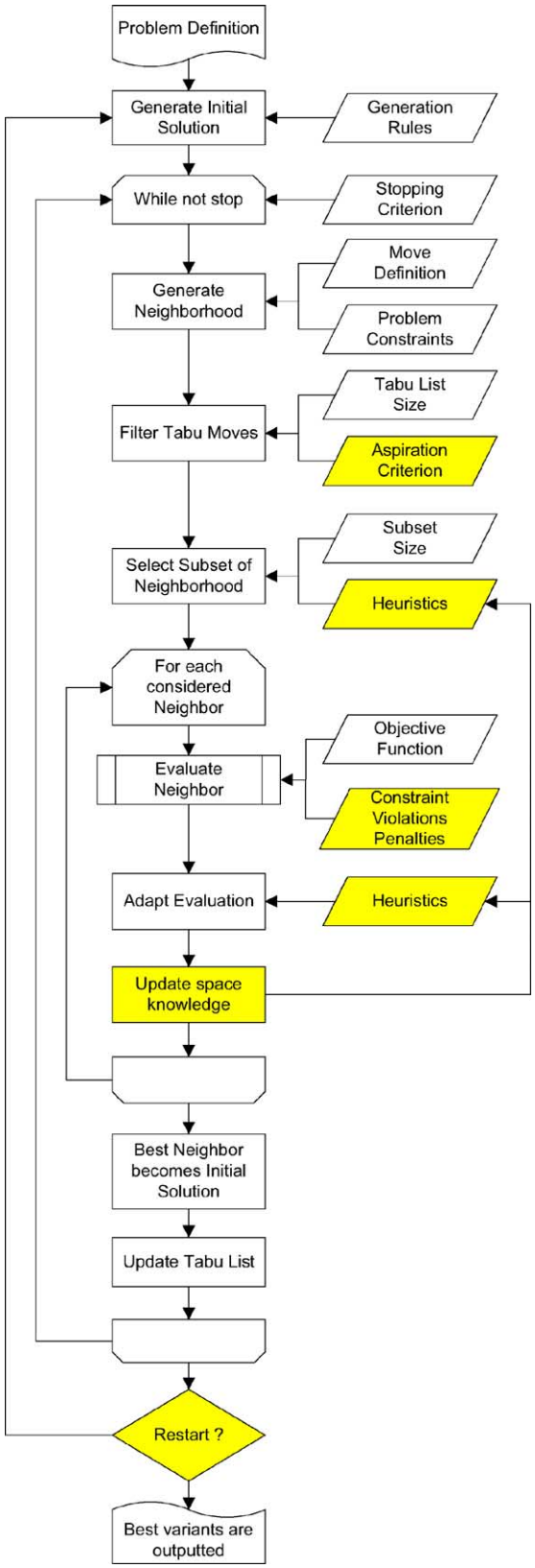

Fig. 3. Tabu Search algorithm. Boxes with gray background signify options. Parallelograms represent the rules and the objective functions.

3.1.3. Neighborhood—candidate list selection

If the moves defined produce a large neighborhood, and if at each iteration every neighbor is evaluated, the algorithm will become quite slow. Therefore, usually only a subset of the neighborhood is considered, produced by examining a candidate list of moves. A straightforward method is to randomly select a fixed number of neighbors for consideration. The trade-off is then: the larger the subset, the slower the algorithm; but the smaller the subset, the higher the risk of requiring many moves to reach an optimum. Relevant considerations for candidate list construction are likewise discussed in the primary TS reference previously indicated (Glover & Laguna, 1997).

3.1.4. Tabu list

The simplest form of adaptive memory used by Tabu Search consists of creating a tabu list of solution attributes that are changed as a result of making recent moves. An attribute on the list can either be a *FromAttribute*, meaning that it belonged to a solution that was left behind as a result of making a move, or a *ToAttribute*, meaning that it belonged to a new solution created by the move. Attributes of either kind that may appear on the tabu list identify moves that are *tabu*, i.e., that are forbidden to be made. If a FromAttribute is on the list, a move is tabu if it would create a solution containing that attribute, while if a ToAttribute is on the list, a move is tabu if it would drop that attribute from the current solution. In either case, the avoidance of a tabu move will prevent the method from re-visiting an associated solution previously encountered. More generally, the tabu classification will also prevent the method from visiting solutions "related to" solutions previously encountered, due to the fact that different solutions can share certain attributes in common. Thus, the mechanism of defining certain moves to be tabu introduces a certain "vigorous diversity" into the sequence of moves generated.

To avoid eliminating moves that can be beneficial to the search, additional mechanisms are employed. The first is simply to limit the size of the tabu list, or stated differently, to limit the *tabu tenure* of any given attribute, i.e., to limit the number of iterations the attribute is permitted to affect the tabu status of potential moves. (Attributes as well as moves are often called tabu. To be precise, the tabu status of a move depends on rules that may specify that it contains some combination of tabu attributes.)

Consequently, the size of the tabu list (the value of tabu tenure) is an important parameter in Tabu Search. Different lists can be created for different types of attributes, thereby affording the possibility for different tenures for these attributes (see below *Asymmetric Tabu List* in Section 5.4). When tabu lists are used as data structures, they operate as *First-In First-Out* (FIFO) stacks. This is the approach used in our current implementation. As an alternative, when the number of solution attributes is not too large, it can be convenient to use a data structure that records for each attribute the iteration when its tabu tenure will end. Once the current

iteration is larger than this recorded value, the attribute is no longer tabu (see below *Oscillating Tabu Lists* in Section 5.4).

Regardless of the data structure employed, the greater the tabu tenure, the smaller are the chances that the algorithm will loop around to re-visit a previously generated solution. But the greater the tenure, the more limited the search becomes. (Good solutions may be missed because a move leading to them remained tabu for a long time.) This fact motivates a second commonly used mechanism to avoid eliminating a potentially beneficial move—*aspiration criteria* that can allow a move to be accepted in spite of being tabu. An aspiration criterion used in most TS implementations is to allow a tabu move to be accepted if it leads to a solution that is better than any solution found so far.

3.1.5. Objective function—best candidate selection

After the neighborhood has been filtered to eliminate tabu moves and a subset of candidate moves has been selected, each neighbor in the subset is evaluated. The best neighbor (having the highest evaluation) is selected, and becomes the "initial" solution for the next iteration. Often the objective function(s) give the basis for the evaluation, although other considerations can also enter. The objective function value itself can be manipulated before the selection. In highly constrained problems for instance, some constraints can be handled by penalty functions.

More advanced forms of TS also use frequency based memory in order to favor (or penalize) a "direction", for example to favor exploring unvisited areas of the solution space (e.g., moves can be encouraged or discouraged according to whether they introduce attributes that were infrequently or very frequently encountered in previous solutions). These designs provide diversification strategies that allow a better covering of the overall solution space. As previously indicated, diversification can also be pursued with multiple restarts in different regions of the solution space (frequency-based memory can also be useful in such restarting strategies).

3.1.6. Stopping criteria

Since the algorithm does not know when the optimum has been found, an external *stopping criterion* must be set. The simplest form of stopping criterion is a fixed number of iterations or a given computational effort. Obviously, the trade-off is: if the algorithm stops too early, the optimum solution may not be found yet. If the algorithm stops too late, computational time can be wasted. A maximum allowable computational time may be useful for problems where the quality of the solution is secondary to the time needed to find it, but a dynamic stopping criterion is more suitable in most cases: if the algorithm does not succeed in improving the existing solution in a given number of iterations, this gives a strong indication that either the optimum has been found, or that the region of the solution space being explored is not interesting—hence that the algorithm should stop or a diversification process (such as a restarting process) should be initiated. When multiple restarts with different initial solutions are used for diversification, a second stopping criterion can be set that limits the number of times the algorithm can restart. This criterion can be either a fixed number of restarts, or a rule according to the history of the optimization.

3.2. Tabu Search implementation

To solve the optimization problem defined in Section 2, our Tabu Search is implemented as follows: a design solution is an assignment of equipment units to each operation block, with design specifications if needed (i.e. use units in parallel or in series); such a solution is represented by the matrix L in Eqs. (14) and (15). Each block has one or several equipment unit(s) allocated, except the transfer blocks (transfers are explicitly listed in the recipe R and in the block matrix B only if they are mandatory; other transfers may be automatically added in the recipe if imposed by the design). The initial solution is randomly generated from the superstructure presented above.

Fig. 4 demonstrates how the different types of moves gradually alter a design during the Tabu Search optimization. To the left, the normalized throughput is displayed for each design. In the middle column, each row represents a part of the current design, the equipment units assigned to operation blocks 1, 2, 3 and 7, at a given iteration (when multiple units are pictured, the operation is conducted in parallel in these units). The complete designs #1 and #9 are shown schematically to the right; the numbers indicate operation blocks being conducted in the equipment pictured (missing blocks are mandatory transfer blocks). The moves are:

(1) Adding an unused equipment unit to a block, leading to designs "in parallel" (as in Fig. 4, seventh row) or to designs "in series".
(2) Removing an equipment unit from the block (as in Fig. 4, second row).
(3) (Optionally) replacing an assigned equipment unit with an unused equipment unit (as in Fig. 4, third row).

The moves may not lead to solutions that violate any constraints. For instance, the move "Add" can only be applied if the operation may have multiple assignments.

The move "replace" is a composition of the two other moves, e.g. add a free piece of equipment, and remove another allocated piece. This move can favor loops, as even if the addition and the removal are tabu, a replacement can have the same effect and is still allowed (the initial tabu list implemented is move-based rather than attribute-based, see Section 5.4). However, the absence of replacement can lead to locking a particular equipment unit in a particular block. In a block that cannot have multiple equipment units allocated (neither in parallel nor in series), the replacement will be difficult: before "adding" a new equipment unit, the previous one must be "removed". But a block with zero equipment units will produce a process design with zero production. Hence such moves will be only selected when all other moves are tabu.

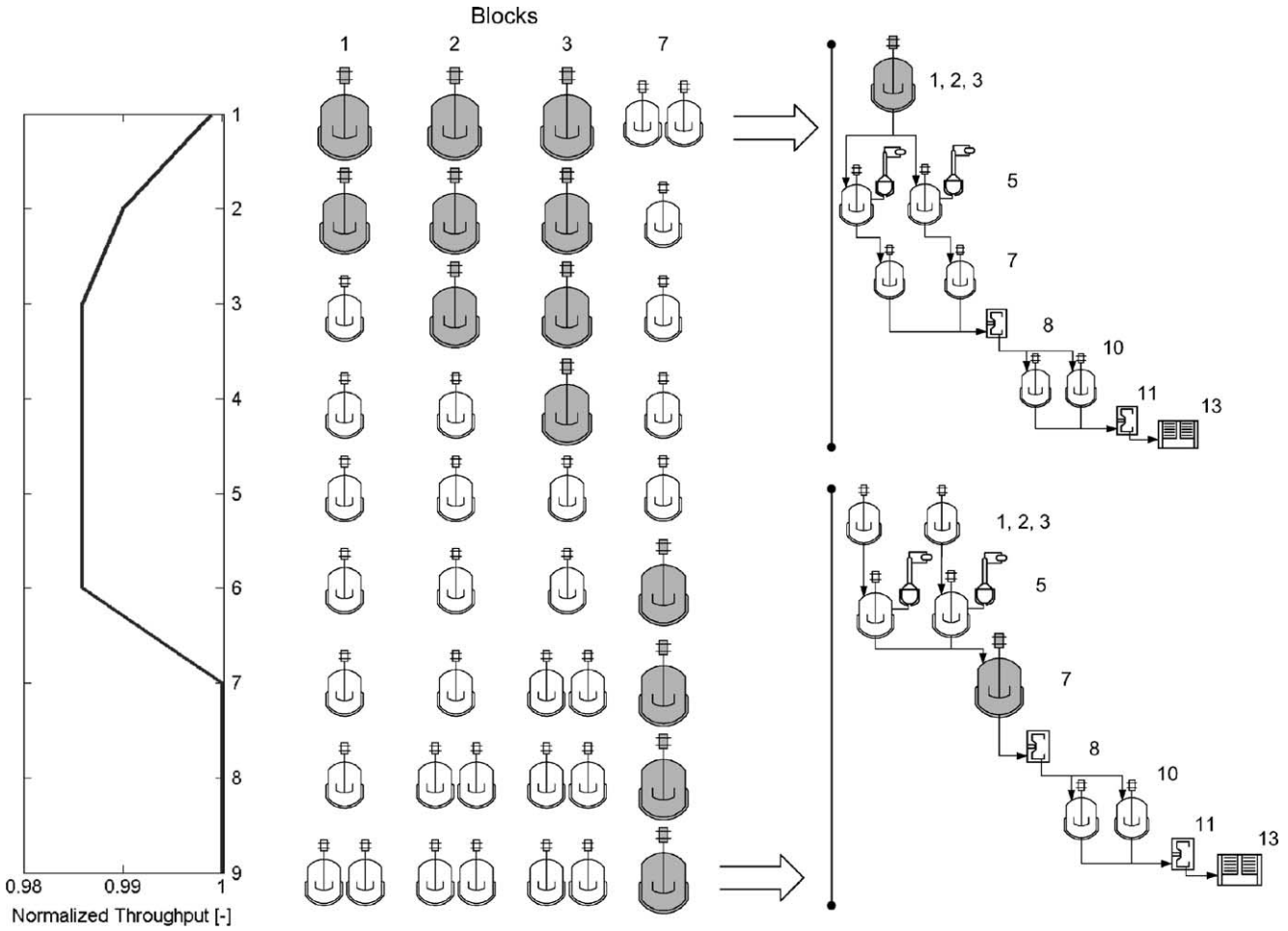

Fig. 4. Demonstration of the different types of moves implemented in the Tabu Search optimization and of the corresponding designs and resulting throughput. The shaded reactor has a volume of $16\,m^3$, while all the others have $10\,m^3$ (see text for further information).

Regarding the options exposed in the introduction above, we chose not to implement any intensification processes, as we focus on a simple type of TS design (as explained earlier). Diversification is achieved solely by means of multiple-restarts, and the stopping criteria consist of (1) a dynamic ending for each restart and (2) a fixed number of restarts.

To determine the parameters that control our implementation, appropriate attention must be given to trade-offs since parameter values that are too high or too low can have a negative impact on the efficiency of the optimization. Section 5 addresses these issues of parameters determination and examines their effect on the performance of the algorithm for the particular class of problems presented here. Different implementations of the tabu list are also investigated in this section.

4. The case studies

Each case study consists of a recipe to be realized and an available plant comprising a set of equipment units. The size of the problem for a particular case study depends on the number of tasks in the recipe as well as on the number of equipment units in the plant. The difficulty of solving a problem depends on its size, and on how constrained it is. If the problem is highly constrained, there will be only few valid designs, hence making the search easier, but the solution space may not be connected, making it impossible to go sequentially from any initial point to the optimum with only valid moves (hence the necessity to have a large number of restarts). If however there are only a few constraints, there will generally be a large number of valid designs and this makes it quite difficult to obtain the global optimum.

Three case studies have been used to illustrate and evaluate the algorithm.

- The first case study involves a rather small and highly constrained problem.
- The second case study involves a rather large, weakly constrained, problem used to demonstrate the scalability of the method.
- The third case study involves a medium size problem, but without any physico-chemical or connectivity

Table 2
Characteristics of the case studies; in the first section, characteristics of the recipe and in the second section, the ones of the production line

	Case Study #1	Case Study #2	Case Study #3
Number of tasks (operations) in recipe	16	93	37
Resulting number of operation blocks	7	33	13
Reaction steps	3	7	2
Number of "transfer" blocks	1	10	4
Maximum number of transfers	5	22	8
Number of equipment units in plant	22	43	31
Number of reactors in plant	7	11	8
Average number of feasible interconnection per equipment unit	9	16	13

constraints. As equipment units are typically standardized in multi-purpose batch plants, this means that many equipment units can be allocated to many blocks and hence that many designs will be equivalent, at least in terms of primary and secondary objective functions. This renders the solution space quite flat and makes it difficult to identify a direction in which the designs improve.

The characteristics of the three case studies are summarized in Table 2. The number of "transfer" blocks indicates how many transfers are required. The maximum number of transfers possible is reached when each equipment unit is allocated to only one block. The number of reactors in a plant is given because reactors are quite versatile and can be used for many classes of operations, in particular when they are equipped with additional equipment (e.g. a distillation can be run in a reactor equipped with a condenser).

In the following, we will compare the results of optimization with the "global optimum". In the rather small Case Study #1, the global optimum is known by proving that no improvements are possible (see Cavin, 2002). In Case Studies #2 and #3, the global optimum is not known. However, thousands of optimization campaigns have been conducted for each case study in the process of setting algorithm parameters, and a high confidence has been reached that the best design found is indeed the global optimum.

5. Results and discussion

5.1. Optimizing the TS parameter settings

We have implemented a Monte Carlo analysis (Vose, 1996) of the algorithm parameters and of some options using the three case studies. The tabu parameters discussed in Section 3.2 (including the decision parameters like "include replacement move") have been randomly set and the problems have been repeatedly optimized using the resulting tabu algorithm. The parameter values have been varied in the following ranges: replacement move and aspiration on or off; tabu list length from 0 to 300; neighbor sample size from 1 to 100; number of process simulations without improvement before restart from 10 to 100; number of restarts from 0 to 20.

This discloses the impact of the different parameters on the efficiency of the algorithm, taking into consideration potential parameter interactions. However, the overall efficiency is much smaller than when only "good" values are used for all parameters at the same time. In a first approximation, only the primary objective function (throughput) is considered. In the next section, all three objectives are included and some differences are highlighted.

5.1.1. Move definition—replacement move

Fig. 5 shows the cumulative distribution functions for runs with and without replacement moves for Case Study #1. The distributions display the probability (vertical axis) that the optimization results in a solution with at least the normalized throughput (i.e. divided by the globally optimum throughput) given on the horizontal axis. The inclusion of the replacement move is clearly favorable: the dotted line is always below the solid line; hence the probability of obtaining a higher production is always larger with the replacement move included. The two other case studies confirm this result, and hence the runs without replacement moves have been excluded from all the following investigations.

5.1.2. Tabu list length and aspiration

The tabu list is managed as a FIFO stack. The size of the tabu list represents therefore how many moves are tabu at a given time and how long a move remains tabu. On Fig. 6, the effect of the tabu list length on the probability to find the optimum throughput is displayed for Case Study #1. We used the "improved best solution" aspiration criterion. The comparison of the probabilities obtained with this aspiration criterion and those without clearly suggests that the inclusion of the criterion is favorable for finding the optimum throughput. As expected, without using an aspiration criterion, the probability decreases with increasing list length. This is due to the "blocking effect" explained above, which is partly countered by the presence of aspiration: the probability rises for small sizes, reaches a summit for sizes around 100 and drops drastically for higher values.

From the outcomes shown in Fig. 6 and similar outcomes obtained in the other case studies, we concluded that aspiration should be enabled and have done this in the following investigations. In view of the sudden drop in probability for

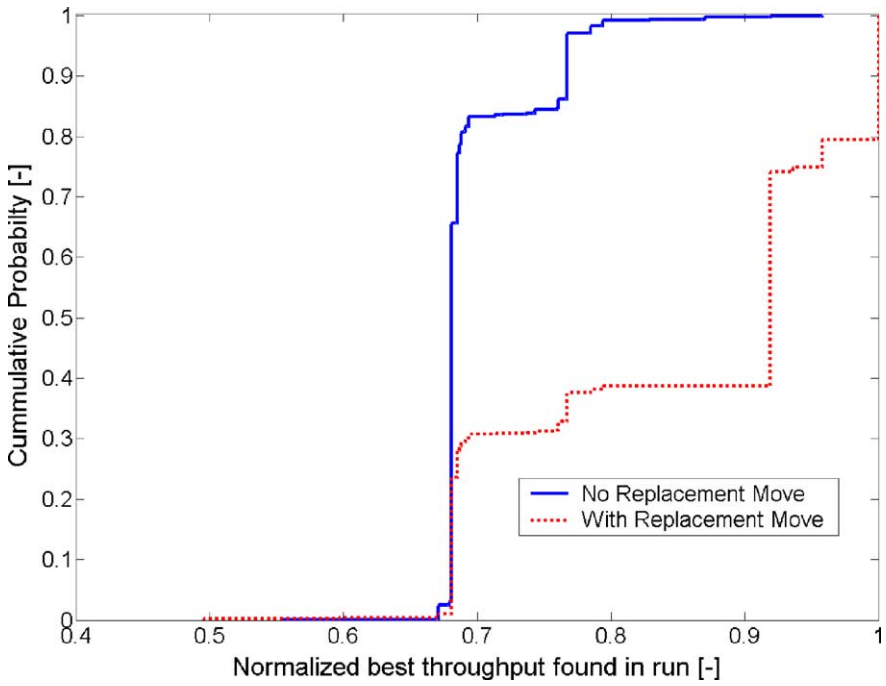

Fig. 5. Cumulative distribution functions for runs with and without the replacement move. The results where obtained for Case Study #1.

list sizes around 100 as displayed in Fig. 6 for Case Study #1, as well as similar findings obtained for the other two case studies, we chose to use a default value of 80 for the tabu list length.

5.1.3. Candidate list strategy and neighbor sample size

We elected to use a straightforward candidate list strategy that randomly samples the neighbors of a given solution. The sample size parameter indicates how many neighbors will be

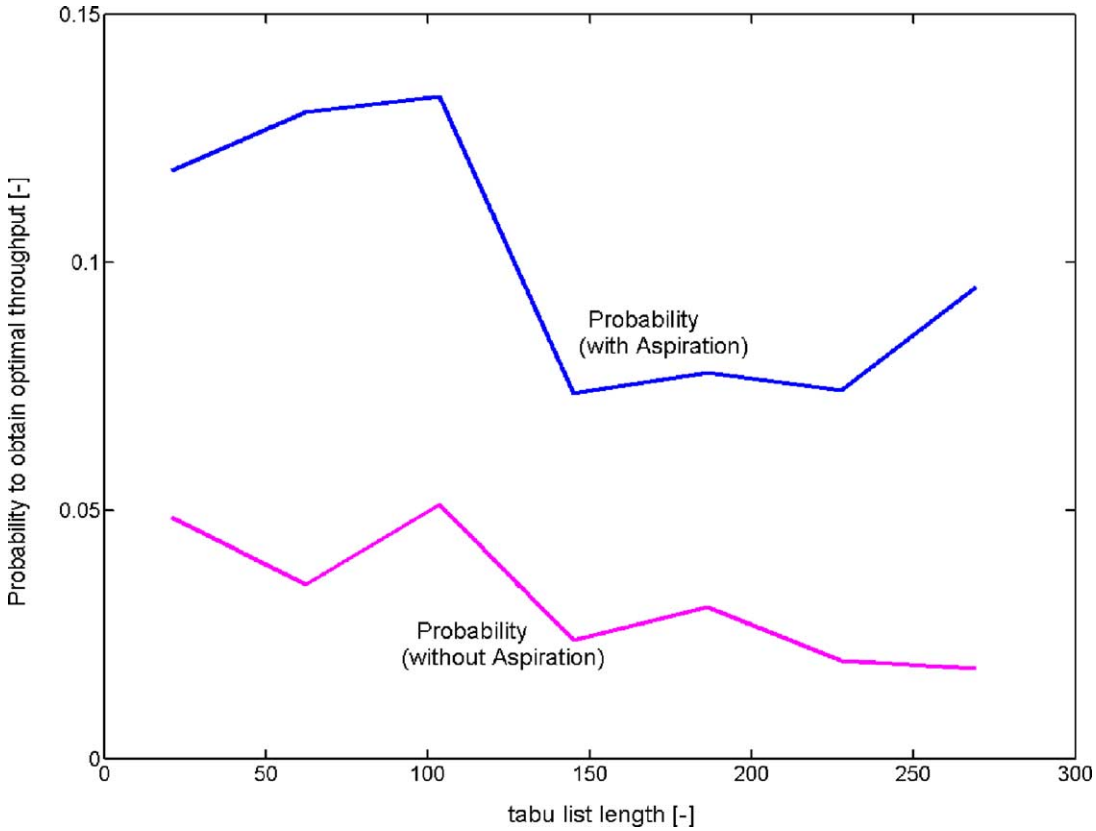

Fig. 6. Probability to obtain the optimum throughput as function of the tabu list length and the inclusion of aspiration for Case Study #1.

investigated per iteration (i.e. the size of the neighborhood's subset as defined above). The investigation of Case Study #1 showed a quite low probability of finding the optimum throughput for small (<5) or large (>25) neighbor sample sizes. For neighbor sample sizes of about 16 the highest probability was found.

The third case study—whose dimensions are slightly larger than Case Study #1—shows a similar trend, with the highest probability at sample sizes of about 20. The second case study—the largest—displays a different trend: the probability rises and stays high for a wide range of neighborhood sizes and decreases for neighborhood sample sizes higher than 45. The fact that Case Study #2 is a large problem is likely the reason that more neighbors are required: the neighborhood sample size should probably be proportional to the size of the problem (as defined by the number of equipment units and the number of blocks), as shown in Fig. 7. Further research would be needed to confirm the relationship indicated in Fig. 7. (The use of more strategic types of candidate list strategies can of course affect this relationship.)

A value of 20 for the sample size was selected as a default for subsequent investigation. This value seems to be a good compromise for the three case studies because the probability to find the optimum is already quite high for this value in Case Study #2.

5.1.4. Stopping strategy

We have found it useful to introduce a measure of computational efficiency in order to investigate the stopping strategy. The probability of finding an optimal solution will always rise (or at least remain constant) when longer runs are conducted—hence to optimize this probability, infinite runs should be conducted. Obviously, the computational effort (represented for instance by the number of process simulations to be conducted) will rise as well with longer runs, hence resulting in a trade-off between probability and computational effort. Therefore, we have defined the *efficiency* of the algorithm as the probability of reaching the optimum throughput, divided by the computational effort invested.

The stopping strategy implemented here is two-fold: first the algorithm stops after a given number of iterations where the solution has not been improved. Then the algorithm restarts with another initial solution. The number of restarts is the second parameter studied.

The investigation of the first parameter shows that whatever its value, the probability of obtaining the optimum throughput is more or less constant; however, the efficiency decreases as the algorithm waits longer. This makes sense as the algorithm will continue the search even in unpromising regions of the solution space.

The investigation of the second parameter, the number of restarts, shows that the probability to find the optimum throughput initially rises sharply with the number of restarts, then rises only slowly for a number of restarts higher than 8. In all three case studies, the efficiency peaks with five to six restarts. This type of behavior again seems reasonable: the more restarts, the more computational effort must be invested, and this occurs even if the global optimum has already been found. But not to restart implies no diversification is employed, and hence there is a great dependency on the initial solution.

Both stopping criteria values may be different if the three objective functions are considered, as subsequent moves may

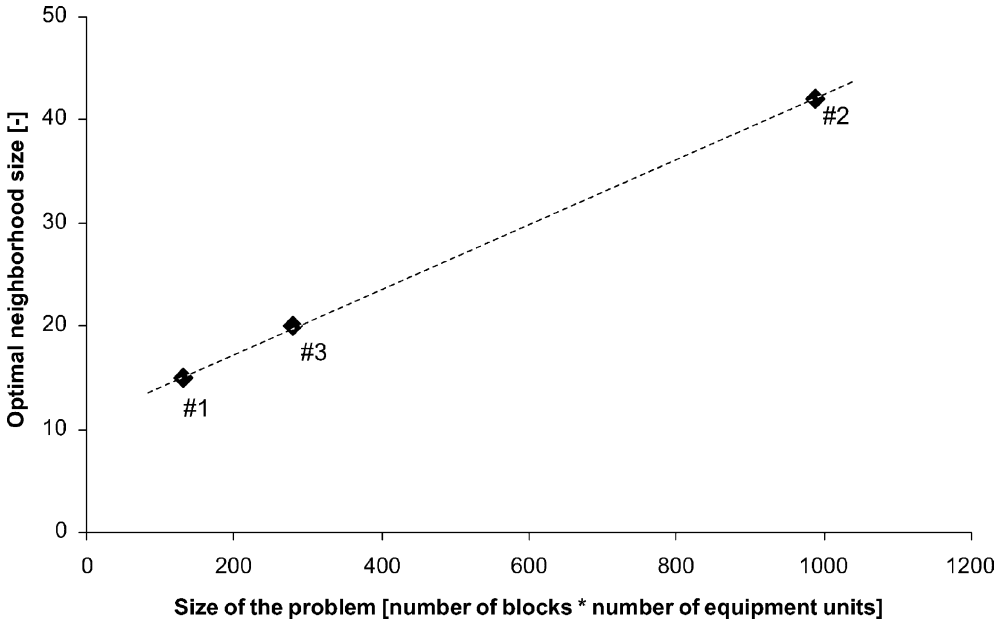

Fig. 7. Optimum neighborhood sample size as a function of problem size and corresponding tentative linear regression; the numbers indicate the corresponding case studies.

be used to optimize the lower priority objective functions. Hence before definitive conclusions on these parameter settings are drawn, the multi-objective optimization problem is investigated.

5.2. Multi-objective optimization

The behavior described thus far arises for the situation where only the main objective function (throughput) is optimized. However, the complete form of the problem has three objective functions: throughput, number of equipment units used (simplicity), and number of floors to pump the reaction mixture up (top–down indicator). When tests are performed by considering all three objective functions together, the conclusions for the parameters do not change noticeably, except for the stopping criteria.

In Fig. 8, for a typical run, the values of the three objective functions are displayed at several iterations after encountering the optimum throughput. The optimization procedure, including some uphill moves, e.g. at point (1), to escape local minima, is still going on in order to improve the secondary and tertiary objective function values. Since the algorithm has no proof of optimality, and continues to seek improvement of all three objectives, uphill moves with regard to the first objective are still possible. Between points (2) and (3) in Fig. 8 no local optimum is found: the algorithm needed to run 25 iterations without improvement to find the global optimum—the optimum with regard to all three objective functions, point (3).

Due to the priority of keeping good higher ranked objective functions values, the secondary and tertiary objective functions are somewhat harder to optimize than the primary. Hence longer runs are needed. This can be seen in Fig. 9, where the stopping criteria have been investigated considering one or all objective functions.

In Case Study #1, no difference was found between taking into consideration only the throughput and taking also into consideration the secondary objective function, indicating that in no runs the optimum throughput has been found without also finding the optimum number of equipment units.

For higher values of the number of iterations without improvement, the efficiency decreases (see Fig. 9a). The highest efficiency occurs for the lowest values when only the throughput is considered. However, when all three objective functions are considered, the greatest efficiency occurs for intermediate parameter values of about 30–40. In addition, the efficiency decreases for a large number of restarts when only throughput is considered (see Fig. 9b), but the efficiency remains high when all three objectives are considered, indicating in this case that the probability of finding the optimum rises in parallel with the computational effort. In sum, about six restarts is a good default value to achieve high efficiencies, although longer runs can be conducted by stipulating a higher number of restarts to augment the probability of finding the global optimum.

5.3. Final parameter settings and efficiency

Table 3 gives the parameter values identified as optimal in the investigations described above for the different case studies, as well as the values that have been chosen as defaults. In the following, we discuss the efficiency of the algorithm in relation to these values.

Table 4 gives a summary of the efficiency of the Tabu Search algorithm determined by the parameter settings as defined in Table 3 for the three case studies.

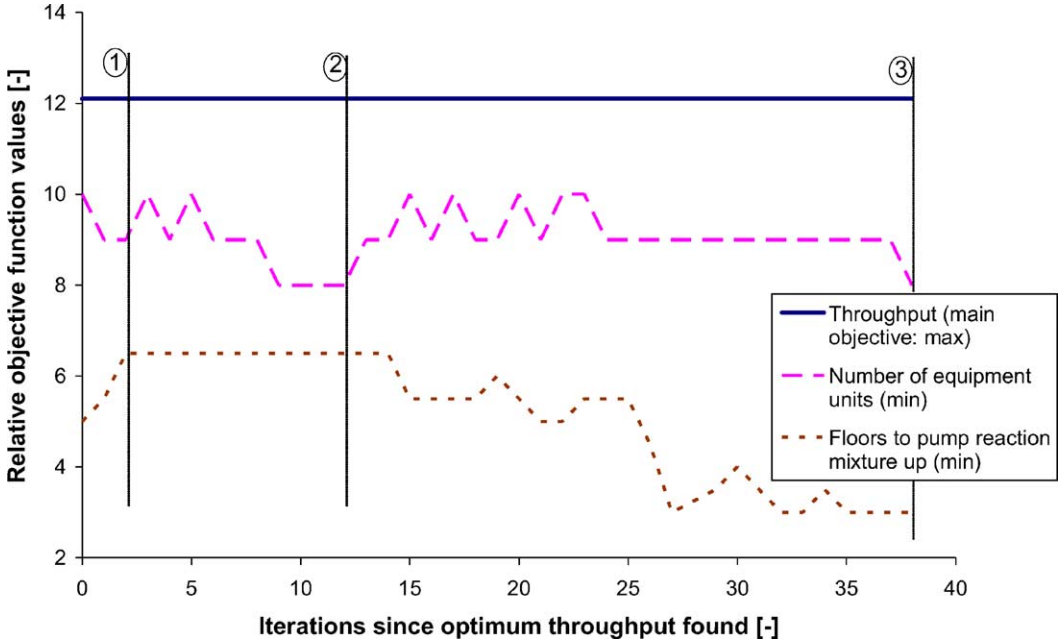

Fig. 8. Values of the arbitrary normalized objective functions during the optimization for Case Study #3.

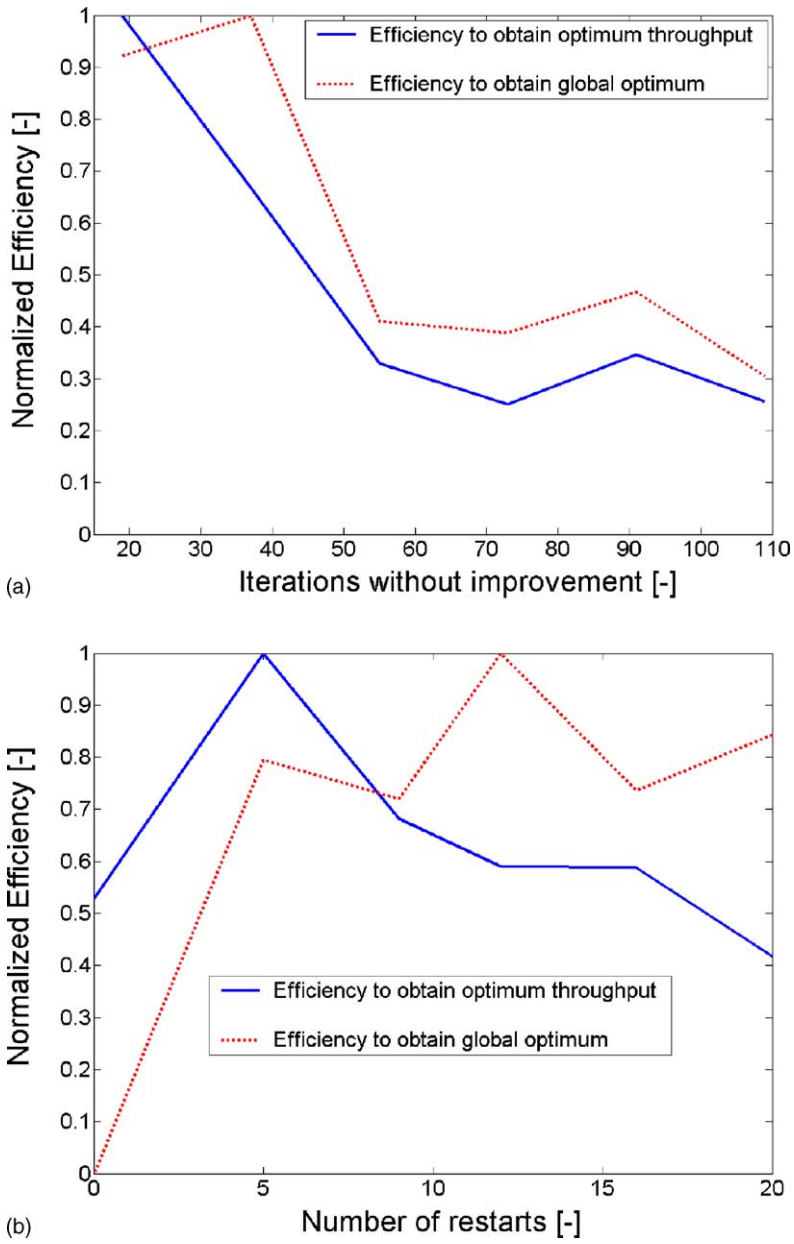

Fig. 9. Algorithm efficiency (see text for definition) as a function of stopping criteria for Case Study #1: (a) number of iterations without improvement before a restart and (b) number of restarts, considering either only throughput or all three objective functions.

Table 3
Optimum parameter values identified for the three case studies (CS) and default values selected

	CS#1	CS#2	CS#3	Default
Aspiration	ON	ON	ON	ON
Replacement move (extended move set)	ON	ON	ON	ON
Tabu list length	95	80	80	80
Neighborhood sample size	15	42	18	20
Iterations without ameliorations before restarting	35	45	40	40
Number of restarts	5	5	6	6

Table 4
Probabilities to find optimum throughput, optimum throughput *and* minimum number of equipment units, and global optimum as obtained for the three case studies (CS) and the parameter settings defined in Table 3

	CS#1 (%)	CS#2 (%)	CS#3 (%)
Probability to find optimum throughput	61.1	97.3	61.8
Probability to find optimum throughput *and* minimum number of equipment units	61.1	90.9	59.5
Probability to find global optimum (according to all three objective functions)	25.2	0.7	11.8

Table 5
Success probability as function of the computational effort for Case Study #1

Computational effort (number of process simulations)	1500	7500	45000
Approximate computational time (min)[a]	2	10	60
Probability of finding optimum throughput (%)	56	98	100[b]
Probability of finding global optimum (with regards to the three objective functions) (%)	25	76	100[b]

[a] Runs conducted on a 1.7 GHz PC with 256 MB RAM, Windows XP®, MATLAB® 6.1.0; no external simulation software has been used, time adaptations have been conducted in MATLAB®.

[b] The probabilities are based on 100 runs. A probability of 100% indicates that all runs found the optimum. The probabilities *not* to find the optimum, as extrapolated from the first two columns are 2×10^{-11} for the optimum throughput, and 2×10^{-4} for the three objective functions.

Due to its constrained nature CS#1 is the most difficult to optimize with regard to the higher ranked objective functions. However CS#2 is quite hard to optimize with regard to the third objective. In this instance many designs have the same throughput and use the same number of equipment units, and hence the optimization of the third objective still takes place over a relatively large part of the solution space.

As mentioned in Section 5.2, to increase the probability of finding optimal designs the number of restarts can be raised. The computational effort will obviously augment in parallel. In Table 5 the performance of the algorithm as a function of the computational effort is given in number of process simulations per run and also in CPU-time for Case Study #1. The probability of finding the optimum throughput and of obtaining a global optimum both rise drastically when longer runs are conducted, reaching a quasi certainty for 1 h CPU.

Similarly, the probability of finding the optimum throughput for Case Studies #2 and #3 rises to at least 99% in 10 min CPU. The probability of finding the global optimum design for Case Study #2—the hardest optimization investigated in this paper—rises from 0.7 to 19% when making 1 h runs, and to about 90% for 10 h runs.

5.4. Tabu list variants

One recurrent problem in the optimization is that the replace operation—while necessary to reach the global optimum—may lock the algorithm into a region around a particular local optimum. If one operation can indifferently use n equivalent equipment units, $n(n-1)$ replacement moves can take place (with any of these n units the throughput and the number of equipment units used remain constant, and only the third objective may vary). If the tabu list is shorter than $n(n-1)$, the algorithm will stay in the same region of space and only a random chance (if the number of available neighbors is significantly higher than $n(n-1)$) will allow to break the pseudo loops by happening not to select a replacement move in the current random neighborhood sample.

This is the reason why the tabu list length (i.e., the tabu tenure) must be so large. As discussed above, a larger tenure removes flexibility and hence makes it hard to find good solutions. To address this problem, some modifications of the tabu lists have been assessed.

5.4.1. Oscillating list length

Large tabu tenures are required to effectively escape a local optimum. However, large tenures hinder the efficient search for a new optimum when the escape has been successful. An interesting option (Glover & Laguna, 1997) consists in activating the tabu classification only upon reaching a local optimum, at which point a large tabu tenure T is enforced. The method continues for T iterations (during which each attribute that becomes tabu remains tabu) or until no non-tabu moves exist. At such a point, the tabu list is emptied and the process once again proceeds freely to a new local optimum.

5.4.2. Strict and asymmetric tabu list

The standard tabu list as implemented in the preceding tests is move-based rather than attribute-based, and forbids the exact opposite of the move accepted. For instance, if in the operation *Op1* the equipment unit *Eq1* is replaced by the unit *Eq2*, the replacement of *Eq2* by *Eq1* in *Op1* will be tabu. As noted in Glover (1990) such move-based approaches entail some risks (including the possibility of being unable to avoid cycling). From an attribute-based perspective, a related option is to specify a move that replaces *Eq2* by *Eq1* in *Op1* to be tabu if the attributes *Eq2* and *Eq1* are themselves both tabu. Still stronger is to make the move tabu if either of its attributes is tabu, hence excluding the removal of *Eq1* from *Op1* and the addition of *Eq2* to *Op1*. (This is clearly more restrictive since, for example, the subsequent replacement of *Eq2* with *Eq3* in *Op1* would now also be forbidden.)

Moreover different types of tabu attributes can be given different tabu tenures, as (for example) by including them on separate lists of differing length. If a list for ‘equipment added’ is longer than a list for ‘equipment removed’, this would block the addition of a previously removed equipment unit longer than the removal of an added one, hence allowing the assessment of new units for the given operation while restricting efficiently the replacement loops described above. If, on the other hand, the removal tabu list is longer, this would still restrict the replacement loops, but favor the emergence of more complex designs with many equipment units per operation (in series or in parallel).

We have tested these options (which can also be combined) on the three case studies. The alternatives that proved interesting are: (a) base case (single tabu list with aspiration), (b) oscillating single tabu list with aspiration, and (c) double asymmetric tabu list, oscillating and with aspiration. The length of the tabu list(s) has been optimized similarly to the other parameters above, and the resulting performances are given in Table 6—similarly as for the results presented

Table 6
Performance of the algorithm for the different tabu list options (based on 1000 runs)

Case study, objective	Base case (%)	Oscillating (%)	Asymmetric and oscillating (%)
CS#1, throughput	61.1	63.4	70.3
CS#1, all objectives	25.2	27.9	12.8
CS#2, throughput	97.3	98.0	98.6
CS#2, all objectives	0.7	0.1	0.6
CS#3, throughput	61.8	63.2	57.1
CS#3, all objectives	11.8	12.7	10.4

in Table 4, the runs have stopping criteria and need about 2 min CPU.

With the exception of CS#2 while considering all three objective functions—whose small success rate makes the 1000 runs sample statistically insufficient to extract conclusions—the oscillating tabu list is slightly better. The runs with asymmetric oscillating tabu lists are significantly better as compared to the base case for CS#1 (optimized for throughput), while being less good for CS#2 and for CS#1 (with all three objectives) and about equivalent for CS#3.

This behavior can be explained by the nature of the case studies: in highly constrained cases (CS#1), complex designs with multiple units in series or parallel are difficult to obtain, and hence the effect given above (with a longer "removal" list) plays a positive role. However the third objective is mainly optimized through replacements (which allow the throughput and number of equipment units to remain constant) and the presence of the strict tabu lists hinder these moves, hence the lower performances with regard to all three objective functions. Similarly, in less constrained cases (CS#3), complex designs do not need to be "encouraged", and the "blocking effect" is too strong.

These results suggest that the well-tuned algorithm might be slightly improved by further refinements. However, similarly to the third tabu list definition (asymmetric double lists), further refinements risk to jeopardize the overall broad applicability of the method: options implemented to tackle-specific problems (like encouraging complex designs) may have a positive effect on some problems but are expected to have a negative effect on others.

5.5. Tabu Search versus multi-start steepest descent

One of the simplest forms of local search is steepest descent, which can also be adapted to our problem. As noted earlier, multi-start (or iterated) steepest descent (mSD) has been recently demonstrated to have appealing theoretical and empirical performance properties, and consequently we have undertaken a comparison with this approach to assess the performance of our basic TS implementation.

As the problem considered is an assignment problem with integer variables and an implicit objective function (black-box), derivatives can not be computed. Therefore, the mSD will simply evaluate all the neighbors, and will select the best one. When all neighbors are worse than the current solution, the descent stops, and a new starting point is randomly chosen.

The comparison is conducted on the basis of the probability to find the optimum throughput for a given computational effort. Obviously, a given application of mSD is much quicker than Tabu Search since it stops upon reaching a first local optimum, and hence many more restarts are possible with this method in the same computational time (factor of 10–20). We based our computation on the Tabu Search approach characterized above in Table 3, with the standard tabu list definition (however, the stopping criterion "number of restarts" was replaced by a maximum number of process simulations, which is roughly equivalent to the computational effort).

In Case Study #1, the mSD finds the optimum throughput in only about 10% of the runs, as compared with the 56% success rate of the Tabu Search as shown in Fig. 10a. These results can be explained by two reasons:

1. The class of problem studied has many local optima and the definition of the moves make uphill moves necessary to strongly modify a solution: for instance to go from a parallel design to a series design, the algorithm must go through a single-unit design, which will in most cases be less efficient; in Fig. 4 the path from the second best design to the optimum also needs uphill moves in order to "exchange" an equipment unit from one block (1) to another (7), as can be seen with the throughput shown on the left side.
2. The constraints forbid some parts of the space, and going "around" the forbidden areas requires most often uphill moves as well.

The second case study, with its only weak constraints, is more suited for the mSD descent as can be seen in Fig. 10b. The TS is still dominant in having a higher probability of finding the optimum throughput (57% against 42% for multi-start steepest descent). However, due to the short duration of the selected runs (cut-off when 2000 process simulations have been computed, about 2 min), the TS profits only from a limited diversification by restarts (4.8 times on average). In contrast, the numerous restarts (more than 80) allowed in the mSD application made it possible to reduce the probability of obtaining bad results (i.e., of obtaining solutions that are not within 90% of the optimum) thanks to the much higher diversification.

If longer runs are selected (e.g. 10 min), many more restarts will be completed by the TS as competitive designs will be memorized, allowing a quicker stopping in uninteresting regions of the search space, resulting in about 46 restarts for the TS versus about 400 for the mSD (a difference of factor 8.8 as compared to 17 in the 2 min runs). The diversification becomes sufficient to avoid completely bad results in the TS as well as in the mSD and results in a complete dominance of TS over mSD similar to the one displayed in Fig. 10a (with much higher performances for

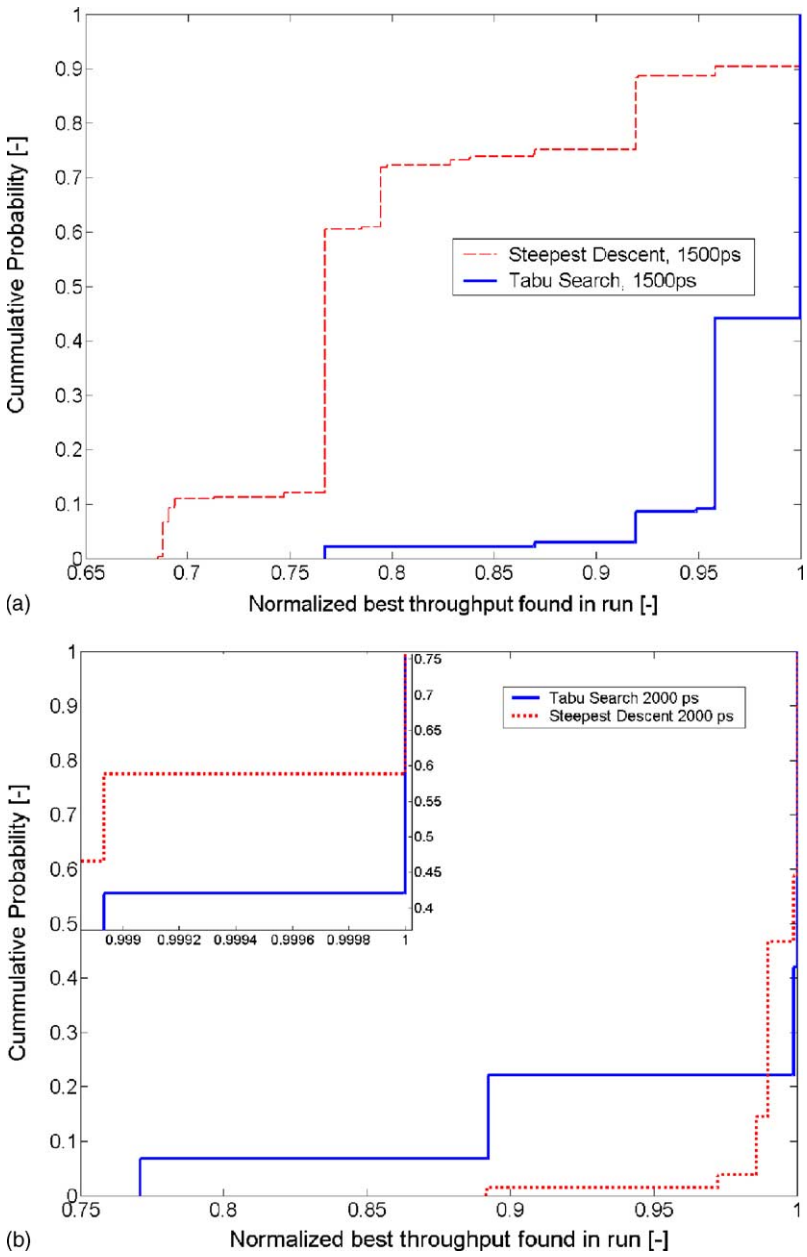

Fig. 10. Comparison of our basic Tabu Search and a multi-start steepest descent for fixed computational effort in: (a) Case Study #1 and (b) Case Study #2, where the top-left shows a detail of the plot for high production rates.

both algorithms). A similar effect is also obtained for the third case study.

6. Conclusions

Our study has focused on a new method for optimizing the implementation of a new single chemical process in a multi-purpose batch plant, in which a well-defined set of equipment units is available. The optimization considers three objectives with different priorities. Our approach embodies a specialized version of Tabu Search, a flexible metaheuristic algorithm, to solve this multi-objective optimization problem. A thorough investigation of algorithm parameters and variants has been conducted to identify combinations that insure a high probability of finding a global optimum while using a minimum of computational effort.

The results obtained for three case studies show that the algorithm is well suited for solving the problems at hand: the resulting probability of obtaining the best throughput within 10 min CPU was higher than 98% for typical problems. When 1 h of CPU-time was allotted, the probability of obtaining the optimum throughput exceeded 99% for all three case studies. Moreover, upon addressing the more challenging multi-objective model, the probability of finding a global optimum within 1 h remained above 97% for

typical problem instances. Computational comparisons with a multi-start steepest descent method, which represents a solution approach recently documented to have desirable performance characteristics, discloses that our Tabu Search method yields superior outcomes in all cases.

The version of TS that has produced these results has been restricted to incorporate only a set of first level elements. Even at this level we have been able to identify robust parameter values, so that the user will not need to modify internal parameters of the optimization algorithm.

We single out several specific components of Tabu Search that we anticipate will provide a useful focus for future investigation:

- An extension of the move set (e.g. inclusion of a move that "exchange" equipment units of two operations) might be profitable in regard of the performance of the "replace" move; it may be useful to implement advanced candidate list strategies to manage the increased neighborhood size. In addition, a strategic oscillation coordinated with alternative neighborhood structures might be interesting.
- Seeing the effectiveness of the simple diversification implemented here (multiple restarts), a promising feature is certainly the integration of intensification and diversification strategies, e.g. with frequency-based memory.

Each of these TS components can be implemented by direct extension of our basic design, utilizing considerations of the type described by Glover and Laguna (1997). These components will have to be tested while focusing on keeping the broad scope of the method. The modification of the tabu list definition has shown that some improvements may jeopardize this, having a positive effect on some case studies but being expected to have a negative impact on others.

The successes obtained by our fundamental TS implementation support an expectation that such an extended approach will nevertheless provide still better results, and make it possible to efficiently solve multiple objective chemical process design problems of still larger dimensions.

References

Cavin, L. (2002). *A systematic approach for multi-objective process design in multi-purpose batch plants* (ISBN 3-906734-31-5). Ph.D. Thesis ETHZ #14898. http://e-collection.ethbib.ethz.ch/show?type=diss&nr=14898.

Chaudhuri, P., & Diwekar, M. (1996). Process synthesis under uncertainty: A penalty Function Approach. *AIChE Journal, 42–43*, 742–752.

Ciric, A. R., & Floudas, C. A. (1990). A mixed integer nonlinear programming model for retrofitting heat-exchanger networks. *Industrial Engineering and Chemical Research, 29*, 239–251.

Date, C. J. (1995). *An introduction to database systems* (6th ed.). New York: Addison-Wesley.

Douglas, J. M. (1985). A hierarchical decision procedure for process synthesis. *AIChE Journal, 31–33*, 353–362.

Floquet, P., Pibouleau, L., & Domenech, S. (1994). Separation sequence synthesis: How to use simulated annealing procedure? *Computers and Chemical Engineering, 18*, 1141.

Fraga, E. S., & Matias, T. R. (1996). Synthesis and optimization of a non-ideal distillation systems using parallel genetic algorithm. *Computers and Chemical Engineering, 20*, S79.

Glover, F., & Hanafi, S. (2002). Tabu Search and finite convergence. *Discrete Applied Mathematics, 119–121*, 3–36.

Glover, F., & Laguna, M. (1997). *Tabu Search*. Dordrecht (Hingham, MA): Kluwer Academic Publishers.

Glover, F. (1990). Tabu Search. Part II. *ORSA Journal on Computing, 2–1*, 4–32.

& Glover, F. (1977). Heuristics for integer programming using surrogate constraints. *Decision Science, 8*, 156.

Gross, B., & Roosen, P. (1998). Total process optimization in chemical engineering with evolutionary algorithms. *Computers and Chemical Engineering, 22*, S229–S236.

Grossmann, I. E., Caballero, J. A., & Yeomans, H. (1999). Mathematical programming approaches to the synthesis of chemical process systems. *Korean Journal of Chemical Engineering, 16*(4), 407–426.

Grossmann, I. E., & Daichendt, M. (1996). New trends in optimization-based approaches to process synthesis. *Computers and Chemical Engineering, 20–26*, 665–683.

Grossmann, I. E., & Kravanja, Z. (1995). Mixed-integer nonlinear-programming techniques for process systems-engineering. *Computers and Chemical Engineering, 19*, S189–S204.

Grossmann, I. E. (1985). Mixed-integer programming approach for the synthesis of integrated process flowsheets. *Computers and Chemical Engineering, 9*, 463.

Grossmann, I. E., & Sargent, R. W. H. (1979). Optimum design of multi-purpose chemical plants. *Industrial Engineering and Chemical Research Development, 18*, 343.

Gruhn, G., & Noronha, S. (1998). Opportunities and limitations of mathematical optimization methods in process synthesis. *Chemical Engineering Technology, 21–28*, 637–640.

Hostrup, M., Gani, R., Kravanja, Z., Sorsak, A., & Grossmann, I. E. (2001). Integration of thermodynamic insights and MINLP optimization for the synthesis, design and analysis of process flowsheets. *Computers and Chemical Engineering, 25*, 73–83.

Jacobson, S. H. (2002). Analyzing the performance of local search algorithms using Generalized Hill Climbing Algorithms. In P. Hansen & C. C. Ribeiro (Eds.), *Essays and surveys on metaheuristics* (pp. 441–467). Dordrecht (Hingham, MA): Kluwer Academic Publishers.

Kallrath, J. (2000). Mixed integer optimization in the chemical industry. Experience, potential and future perspectives. *TransIChemE, 78*(A6), 809–822.

Kirkpatrick, S., Gelatt Jr., C.D., & Vecchi, M. P. (1983). Optimization by simulated annealing. *Science, 220*, 671–680.

Kirkwood, R. L., Locke, M. H., & Douglas, J. M. (1988). A prototype expert system for synthesizing chemical process flowsheets. *Computers and Chemical Engineering, 12–14*, 329–343.

Loonkar, Y., & Robinson, J. D. (1970). Minimization of capital investment for batch processes. Calculation of optimum equipment sizes. *Industrial Engineering and Process Design Development, 9*, 625.

Lourenco, H. R., Martin, O. C., & Stutzle, T. (2002). Iterated local search. In G. Kochenberger & F. Glover (Eds.), *Handbook of metaheuristics* (pp. 183–197). Dordrecht (Hingham, MA): Kluwer Academic Publishers.

Mauderli, A., & Rippin, D. W. T. (1979). Production planning and scheduling for multi-purpose batch chemical plants. *Computers and Chemical Engineering, 3*, 199.

Papageorgaki, S., & Reklaitis, G. V. (1990). Optimal design of multi-purpose batch plants. 1. Problem formulation. *Industrial Engineering and Chemical Research, 29*, 2054.

Papalexandri, K. P., & Pistikopoulos, E. N. (1996). Generalized modular representation framework for process synthesis. *AIChE Journal, 42–44*, 1010–1032.

Rippin, D. W. T. (1983). Design and operation of multiproduct and multi-purpose batch chemical plants: An analysis of problem structure. *Computers and Chemical Engineering, 7*, 463.

Siirola, J. J. (1996). Strategic process synthesis: Advances in the hierarchical approach. *Computers and Chemical Engineering, 20*, S1637–S1643.

Skrifvars, H., Harjunkoski, I., Westerlund, T., Kravanja, Z., & Porn, R. (1996). Comparison of different MINLP methods applied on certain chemical engineering problems. *Computers and Chemical Engineering, 20*, S333–S338.

Sparrow, R. E., Forder, G. J., & Rippin, D. W. T. (1975). The choice of equipment sizes for multiproduct batch plants. Heuristics vs. branch and bound. *Industrial Engineering and Chemical Process Design Development, 14*, 197.

Suhami, I., & Mah, R. S. H. (1982). Optimal design of multi-purpose batch plants. *Industrial Engineering and Chemical Process Design Development, 21*, 94.

Takamatsu, T., Hashimoto, I., & Hasebe, S. (1982). Optimal design and operation of a batch process with intermediate storage tanks. *Industrial Engineering and Chemical Process Design Development, 21*, 431.

Vose, D. (1996). *Quantitative risk analysis: A guide to Monte-Carlo simulation modeling*. Hoboken: Wiley.

Voudouris, V. T., & Grossmann, I. E. (1992). Mixed-integer linear programming reformulations for batch process design with discrete equipment sizes. *Industrial Engineering and Chemical Research, 31*, 1315.

Wang, C., Quan, H., & Xu, X. (1999). Optimal design of multiproduct batch chemical processes using tabu search. *Computers and Chemical Engineering, 23*, 427.

Yeh, N. C., & Reklaitis, G. V. (1987). Synthesis and sizing of batch/semicontinuous processes: Single product plants. *Computers and Chemical Engineering, 11*, 639.