On dominance-based multiobjective local search: design, implementation and experimental analysis on scheduling and traveling salesman problems

Arnaud Liefooghe, Jérémie Humeau, Salma Mesmoudi, Laetitia Jourdan, El-Ghazali Talbi

Journal of Heuristics manuscript No.
(will be inserted by the editor)

On Dominance-based Multiobjective Local Search: Design, Implementation and Experimental Analysis on Scheduling and Traveling Salesman Problems

Arnaud Liefooghe · Jérémie Humeau · Salma Mesmoudi · Laetitia Jourdan · El-Ghazali Talbi

Received: 07-01-2009 / Accepted: date

Abstract This paper discusses simple local search approaches for approximating the efficient set of multiobjective combinatorial optimization problems. We focus on algorithms defined by a neighborhood structure and a dominance relation that iteratively improve an archive of nondominated solutions. Such methods are referred to as dominance-based multiobjective local search. We first provide a concise overview of existing algorithms, and we propose a model trying to unify them through a fine-grained decomposition. The main problem-independent search components of dominance relation, solution selection, neighborhood exploration and archiving are largely discussed. Then, a number of state-of-the-art and original strategies are experimented on solving a permutation flowshop scheduling problem and a traveling salesman problem, both on a two- and a three-objective formulation. Experimental results and a statistical comparison are reported in the paper, and some directions for future research are highlighted.

Keywords Multiobjective optimization · Local search · Flowshop scheduling problem · Traveling salesman problem

A. Liefooghe · L. Jourdan · E-G. Talbi
Laboratoire d'Informatique Fondamentale de Lille (LIFL), UMR CNRS 8022, Université Lille 1
INRIA Lille-Nord Europe
Cité scientifique, Bât. M3, 59655 Villeneuve d'Ascq cedex, France

A. Liefooghe
Tel.: +33 3 59 57 79 30
Fax: +33 3 59 57 78 50
E-mail: arnaud.liefooghe@univ-lille1.fr

J. Humeau
École des Mines de Douai, Département IA, 941 rue Charles Bourseul, BP 10838, 59508 Douai, France
E-mail: jeremie.humeau@mines-douai.fr

S. Mesmoudi
Laboratoire d'Imagerie Fonctionnelle (LIF), UMR 678 Inserm–UPMC, CHU Pitié-Salpêtrière
91 boulevard de l'Hôpital, 75634 Paris cedex 13, France
E-mail: salma.mesmoudi@iscpif.fr

L. Jourdan
E-mail: laetitia.jourdan@inria.fr

E.-G. Talbi
E-mail: talbi@lifl.fr

1 Introduction

One of the most challenging task in the field of multiobjective optimization relates to the identification of the efficient solution set, or an approximation of it for rather difficult optimization problems. On the one hand, evolutionary algorithms are commonly used to this end, since multiple solutions can be found in a single simulation run. On the other hand, local search methods are known to provide good-quality solutions for many hard combinatorial optimization problems. However, in comparison to the tremendous number of existing multiobjective evolutionary algorithms, the number of multiobjective local search approaches is extremely small. They provide an interesting alternative to classical evolutionary algorithms and often handle a relatively small number of parameters, but they are usually restricted to a role of sub-procedure in the frame of a memetic or hybrid multiobjective metaheuristic. Furthermore, from a practitioner point of view, it is often the case that a well-performing single-objective local search method, with advanced problem-related components, is to be extended in order to solve a multiobjective counterpart of the same problem. Then, designing a multiobjective evolutionary algorithm for its investigation will not allow to reuse much of the components developed for the single-objective problem.We here try to outline the main issues to study before adapting or designing a local search method based on a dominance relation.

In this paper, we focus on a class of pure neighborhood search methods for multiobjective combinatorial optimization. These simple algorithms can be seen as a generalization of the most basic local search procedure (also known as hill-climbing, descent, iterative improvement, etc.) for the multiobjective case. Generally speaking, they combine the definition of a neighborhood structure with the management of an archive of potentially efficient solutions, according to a dominance relation. This archive is iteratively improved by exploring the neighborhood of its own content until no further improvement is possible, or until another stopping condition is satisfied. Such methods are often referred to as *Pareto-based* or *Pareto Local Search* (Paquete et al, 2004; Talbi et al, 2001; Basseur et al, 2002). However, we found the denomination *Dominance-based Multiobjective Local Search* (DMLS) more relevant, because, contrary to existing approaches, other pairwise dominance relations, such as weak-dominance, strict-dominance and so on, can be used to discriminate solutions instead of the classical Pareto-dominance. Besides, the number of DMLS algorithms is very limited in proportion to existing local search approaches dealing with the approximation to the efficient set (Ehrgott and Gandibleux, 2004). Most methods are generally based on the successive resolution of multiple scalarizations of the same objective vector function. Multiobjective local search principles based on a dominance relation appeared more recently in the literature.

State-of-the-art DMLS algorithms are first reviewed. They include the Pareto Local Search (Paquete et al, 2004; Talbi et al, 2001; Basseur et al, 2002), the Biobjective Local Search (Angel et al, 2004), the Pareto Archived Evolution Strategy (Knowles and Corne, 2000), the Simple Evolutionary Multiobjective Optimizer (Laumanns et al, 2004), and a random bit climber proposed by Aguirre and Tanaka (2005). The main issues related to their design are identified, and an attempt of unifying these methods is provided. The problem-free components of dominance relation, current set selection, neighborhood exploration, archiving and stopping condition are described in detail, and a number of strategies for such purpose are largely discussed. Next, we illustrate how the proposed unified view allows to address existing methods as simple variants of the same structure, and how original methods can conveniently be designed. In addition, we propose a software package, integrated into the ParadisEO framework, that is dedicated to the flexible and reusable design of DMLS

algorithms. Finally, a number of strategies for each design step of a DMLS method are experimented with. In particular, up to now, it has remain unclear how to: (*i*) select a proper set of solutions whose neighborhood is to be explored, (*ii*) design a proper neighborhood exploration strategy from this current set. Two combinatorial optimization problems, involving a reasonable number of objective functions, are investigated. They consist of a permutation Flowshop Scheduling Problem (FSP) and of a Traveling Salesman Problem (TSP), each one being formulated as two and three objective problems. The strategies under investigation give rise to a total number of eight DMLS algorithms that are rigorously compared with each other on a set of both FSP and TSP benchmark test instances. The experiments conducted in the paper show that DMLS algorithms are easily scalable, and successfully find a good approximation of the efficient set for different problem types and sizes. We discuss their respective behaviors and we try to highlight some useful guidelines about the choice of the main DMLS-related design issues for multiobjective combinatorial optimization.

The paper is organized as follows. In Section 2, we begin with some background information about multiobjective combinatorial optimization and multiobjective search methods. In Section 3, a model for DMLS is proposed, and its main search components are largely discussed. In Section 4, we provide a comparative study of different DMLS strategies for solving two- and three-objective variants of a FSP and of a TSP. At last, Section 5 concludes the paper and discusses future research.

2 Background Information

This section presents some basic concepts, notation and definitions about multiobjective combinatorial optimization. Next, some issues related to the design of metaheuristics for such purpose as well as a brief review and classification of existing multiobjective local search methods are given.

2.1 Multiobjective Combinatorial Optimization

A *Multiobjective Combinatorial Optimization Problem* (MCOP) aims to optimize a set of $n \geq 2$ objective functions $(f_1, f_2, \ldots, f_n)$ simultaneously. Let X denote the discrete set of feasible solutions in the *decision space* (that usually has some additional combinatorial structure), and Z the set of feasible vectors in the *objective space*. Without loss of generality, we assume that $Z \subseteq \mathscr{R}^n$ and that all n objective functions are to be minimized. To each decision vector $x \in X$ is assigned an objective vector $z \in Z$ on the basis of the vector function $f : X \rightarrow Z$ with $z = f(x)$. A MCOP can be defined as follows:

$$\begin{array}{l} \text{minimize } f(x) = (f_1(x), f_2(x), \ldots, f_n(x)) \\ \text{subject to } x \in X \end{array} \tag{1}$$

A dominance relation is then usually assumed so that a partial order is induced over X. Numerous dominance relations exist in the literature and will be discussed later in the paper. Let us define the well-known concept of *Pareto-dominance*.

Definition 1 (Pareto-dominance) An objective vector $z \in Z$ is said to *dominate* an objective vector $z' \in Z$ iff $\forall i \in \{1,2,\ldots,n\}$, $z_i \leq z'_i$ and $\exists j \in \{1,2,\ldots,n\}$ such that $z_j < z'_j$. This relation will be denoted by $z \succ z'$.

We will also say that a decision vector $x \in X$ *dominates* a decision vector $x' \in X$, denoted by $x \succ x'$, if $f(x)$ dominates $f(x')$.

Definition 2 (Nondominated vector) An objective vector $z \in Z$ is said to be *nondominated* iff there does not exist another objective vector $z' \in Z$ such that $z' \succ z$.

Definition 3 (Efficient solution) A solution $x \in X$ is said to be *efficient* (or *Pareto optimal*, *nondominated*) iff its mapping in the objective space results in a nondominated vector.

The set of all efficient solutions is called *efficient* (or *Pareto optimal*) *set*, denoted by X_E, and its mapping in the objective space is called *Pareto front*, denoted by Z_N. A possible approach in MCOP solving is to find the minimal set of efficient solutions, *i.e.* one solution $x \in X_E$ for each nondominated vector $z \in Z_N$ such that $f(x) = z$ (in case multiple solutions map to the same nondominated vector). But, generating the entire set of Pareto optimal solutions is usually infeasible, due to the complexity of the underlying problem or to the large number of optima. Therefore, the overall goal is often to identify a good efficient set approximation.

2.2 Metaheuristics for Multiobjective Combinatorial Optimization

Metaheuristics are a powerful class of general-purpose search methods (Talbi, 2009). They have been successfully applied to a wide-range of optimization problems, and are able to find good-quality solutions in a reasonable runtime. Approximating an efficient set is itself a bi-objective problem. Indeed, the approximation to be found must have both good convergence and distribution properties, as its mapping in the objective space has to be (*i*) close to, and (*ii*) well-spread over the (generally unknown) optimal Pareto front. As a consequence, the main difference between the design of a single-objective and of a multiobjective search method deals with these two goals. Since they are naturally well-suited to find multiple solutions in a single simulation run, population-based metaheuristics are commonly used to approximate a MCOP efficient set. In particular, a large amount of multiobjective evolutionary algorithms have been proposed in the literature. Over the last decades, major advances, from both algorithmic and theoretical aspects, have been made in the field of evolutionary multiobjective optimization (Deb, 2001; Coello Coello et al, 2007). However, local search methods are known to be effective metaheuristics and to provide high quality solutions for solving many hard real-world applications. Nevertheless, a relatively small number of multiobjective local search algorithms have been proposed so far. They are briefly reviewed in the next section. The reader is referred to (Ehrgott and Gandibleux, 2004) for more information about metaheuristics for multiobjective combinatorial optimization.

2.3 Multiobjective Local Search

A classical local search algorithm, also referred to as hill-climbing, descent, iterative improvement, etc., consists of iteratively improving an arbitrary solution according to its neighborhood until a local optimum is reached. Hence, while dealing with such methods, the definition of a neighborhood structure is required for the problem under consideration.

Definition 4 (Neighborhood structure) A *neighborhood structure* is a mapping function $\mathcal{N} : X \rightarrow 2^X$ that assigns a set of solutions $\mathcal{N}(x) \subset X$ to any solution $x \in X$. $\mathcal{N}(x)$ is called the *neighborhood* of x, and a solution $x' \in \mathcal{N}(x)$ is called a *neighbor* of x.

Given that efficient solutions are to be found in the frame of multiobjective optimization, the notion of local optimum has to be defined in terms of Pareto optimality. Let us define the concept of locally efficient solution.

Definition 5 (Locally efficient solution) A solution $x \in X$ is said to be *locally efficient* with respect to a neighborhood structure $\mathcal{N}$ iff there does not exist a neighboring solution $x' \in \mathcal{N}(x)$ such that $x' \succ x$.

In the field of multiobjective optimization, a neighborhood search algorithm can generally be used as a stand-alone approach to approximate a MCOP efficient set, or either to be hybridized with other methods; see (Ehrgott and Gandibleux, 2008) for an overview of hybrid multiobjective metaheuristics. Initial multiobjective local search approaches were based on the successive and independent improvement of a single solution. Generally speaking, the final approximation to be found is contained into an external set where potentially efficient solutions are stored. However, some recent techniques intrinsically handle a population of solutions to be evolved in parallel. Such methods can then capitalize the knowledge available within the population to guide and focus the search. In practice, combining local search principles with the use of a population appears to be particularly promising for MCOP solving. Thus, while aiming at finding an efficient set approximation, existing multiobjective local search methods are generally divided into two different categories: *scalar approaches* and *dominance-based approaches*. They are briefly described below. More details can be found in (Paquete and Stützle, 2007).

Approaches belonging to the first class consist of solving multiple single-objective optimization problems by means of reducing the original MCOP through the scalarization of the objective function vector. Scalar methods are often based on a weighted-sum aggregation, *i.e.* $f_\lambda(x) = \sum_{i=1}^{n} \lambda_i f_i(x)$, where $\lambda_i > 0$ for all $i \in \{1,\ldots,n\}$. However, in the combinatorial case, note that a number of efficient solutions, known as *non-supported efficient solutions*, are not optimal for any weighted-sum aggregation function. Alternatively, techniques such as ε-constraint or reference point methods can also be applied. The reader is referred to Miettinen (1999) for a detailed description of scalarization approaches for discrete multiobjective optimization. Then, once a scalar approach is defined, say a simple weighted-sum aggregation function f_λ, a classical single solution-based metaheuristic is applied until a local optimum, with respect to f_λ, is reached. This process is iterated by modifying the weight vector λ. For each scalarization, the best found solution is incorporated into the efficient set approximation whose dominated solutions are later discarded. For such purpose, single solution-based metaheuristics go from simple local search (Paquete and Stützle, 2003) to more advanced methods like tabu search (Gandibleux et al, 1997; Hansen, 1997) or simulated annealing (Serafini, 1992; Ulungu et al, 1999).

The second class, which is of our interest in this paper, consists of defining the acceptance criterion for a dominance relation, like Pareto-dominance as given in Def. 1. This class will be denoted by DMLS, for Dominance-based Multiobjective Local Search, in the remainder of the paper. The idea of the most basic DMLS algorithm is to maintain an archive of nondominated solutions, to explore the neighborhood of archive members, and to update the content of the same archive with nondominated neighboring solutions until it does not improve anymore. Therefore, contrary to scalar approaches, the archive is not only used as an external storage, but also corresponds to the population to be improved. Different variants of this basic idea have been proposed in the literature. Firstly, the *Pareto Local Search* (Paquete et al, 2004) consists of starting from an initial solution to be added to the archive. Then, a single solution is randomly selected from the archive. All its neighbors are then evaluated and nondominated ones are proposed as candidate solutions to enter the

archive. These steps are iterated until no further improvement is possible. This algorithm has been proved to converge to a Pareto local optimum set by Paquete et al (2007). Very similar approaches have been proposed (Talbi et al, 2001; Basseur et al, 2002; Angel et al, 2004). The only difference is that the neighborhood is here explored from the whole set of archive members, and not from a single solution. Next, the *Pareto Archived Evolution Strategy* (PAES) (Knowles and Corne, 2000) is a local search evolution strategy where a single random neighbor is evaluated from a single archived solution. But note that PAES variants where more than one archive solution is selected, and where more than one neighbor is generated *per* solution have also been investigated (Knowles and Corne, 2000). Another notable difference is that PAES uses a bounded-size archiving technique to truncate the size of the current nondominated set. A similar idea has been proposed by Laumanns et al (2004) for the SEMO and FEMO algorithms, where a random neighbor of one solution from the archive is explored. Such approaches also served as a basis from numerous theoretical results for multiobjective optimization (Neumann and Wegener, 2006; Brockhoff et al, 2007). At last, different multiobjective local search algorithms have been investigated for bit strings by Aguirre and Tanaka (2005). One of them, referred to as *Random Bit Climber using the Archive for Restarts*, explores the neighborhood of a single archived solution until a dominating one is found, or once no more neighbor can be created. If an improving neighbor is found, it replaces the initial solution in the bounded archive. This process is repeated for all archive members until no more progress can be made. This class of algorithms has shown its efficiency for various MCOPs, either as stand-alone approaches (Paquete et al, 2004; Paquete and Stützle, 2006; Geiger, 2007) or hybridized with other techniques like hybrid metaheuristics (Talbi et al, 2001; Basseur et al, 2002) and two-phase heuristics (Paquete and Stützle, 2006; Paquete and Stützle, 2009; Dubois-Lacoste et al, 2009; Lust and Teghem, 2010; Lust and Jaszkiewicz, 2010).

In the next section, we show that DMLS algorithms share a large number of basic components. A general-purpose model unifying this class of multiobjective metaheuristics is given and some related issues are discussed in detail.

3 A Model for Dominance-based Multiobjective Local Search

Conceiving a generic model is a relevant practice to abstract the design- and the implementation-specific details and to provide a general formulation. Until now, each DMLS algorithm was designed independently of the others, and was implemented as a self-contained method with its own specific elements. In the following, we identify the main search components shared by all DMLS algorithms. Then, a unifying model that takes them into account is presented in an attempt of providing a common terminology and classification. Such a general classification provides a common description and comparison of DMLS algorithms, and will allow to design new methods borrowing ideas from existing ones. Hence, whatever the MCOP to be solved, the main search components for the design of a DMLS algorithm can be stated as follows:

1. Design a representation;
2. Design an initialization strategy;
3. Design a way of evaluating a solution;
4. Design a suitable neighborhood structure;
5. Design a way of evaluating a neighboring solution incrementally (if possible);
6. Decide a dominance relation;
7. Decide a current set selection strategy;

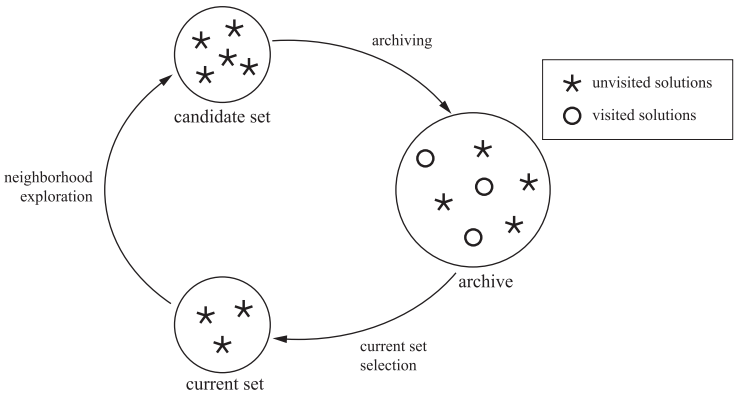

Fig. 1 Model for DMLS algorithms.

8. Decide a neighborhood exploration strategy;
9. Decide an archive management strategy;
10. Decide a stopping condition.

When dealing with any kind of metaheuristics, one may distinguish problem-related and problem-independent components. Hence, the first five issues presented above strongly depend of the MCOP under consideration, whereas the latter five ones can be seen as generic components. In comparison to single-objective optimization, the evaluation and the incremental evaluation can be seen as multiobjective-specific components because multiple objective functions are now to be computed. With regards to problem-free issues, almost all components are explicitly defined for the multiobjective case, except for some simple stopping conditions. Three additional data structures are involved to store (*i*) the current archive content, (*ii*) the *current set* of solutions whose neighborhood is to be explored, and (*iii*) the *candidate set* of neighbor solutions that will potentially enter the archive. Note that, in some particular cases, if the neighborhood of a given solution is evaluated in an exhaustive way, the corresponding solution should be marked as visited in order to avoid a useless neighborhood revaluation. So, the archive may contain both visited and unvisited solutions.

Following this fine-grained decomposition, a conceptual model is presented in Fig. 1. Since problem-related components are assumed to be designed for the MCOP at hand, they do not appear in the figure. The model starts with a user-given set of mutually nondominated solutions. These solutions are used to initialize the archive. Then, the following three steps are iterated until a stopping criteria is satisfied. Firstly, a subset of archive solutions is selected to build the current set. Next, the neighborhood of the current set is explored to build the candidate set. At last, the archive is updated with new solutions from the candidate set. At each step, a specific strategy is to be decided in order to build a specific DMLS instance. The remainder of this section provides a detailed description of the components involved in the DMLS model. Next, some state-of-the-art methods are treated as simple instances of it.

3.1 Problem-related Issues

The design of problem-specific search components is briefly discussed below.

– *Representation*. Solution representation is the starting point for anyone who plans to design any kind of metaheuristic. A MCOP solution needs to be represented both in the

decision space and in the objective space. While the representation in the objective space can be seen as problem-independent, the representation in the decision space must be relevant to the tackled problem.
- *Initialization*. Whatever the algorithmic solution to be designed, an initialization strategy is expected. The way to initialize a solution (or a population of solutions) is closely related to the problem under consideration and to the representation at hand.
- *Evaluation*. The problem at hand is to optimize a set of objective functions simultaneously over a given search space. Then, each time a new solution is produced, its objective vector must be evaluated. Each objective vector component must quantify the quality of the solution under consideration with respect to the corresponding objective function.
- *Neighborhood*. The design of a local search method requires the definition of a *neighborhood* structure for the problem under consideration (see Def. 4). This is a key issue for the local search efficiency. Note that different variants of the same local search can be distinguished with respect to the order in which the neighboring solutions are generated: deterministically, randomly, or also adaptively.
- *Incremental evaluation*. Since evaluating a solution is often the most expensive step of a local search algorithm, an efficient way to evaluate a neighboring solution is rather appreciated, when it is possible to compute it.

3.2 Problem-independent Issues

A detailed description of the five invariant components involved in the DMLS model is given in this section. Additionally, state-of-the-art schemes as well as novel strategies are presented and classified for dominance relation, current set selection, neighborhood exploration, archiving and stopping condition.

3.2.1 Dominance Relation

Within dominance-based approaches for multiobjective optimization, the dominance relation under consideration is generally based on Pareto-dominance as given in Def. 1. However, in the literature, other dominance criteria are found and can easily be decided instead of Pareto-dominance to instantiate a DMLS algorithm. Let us define the following examples.

Definition 6 (Weak-dominance) An objective vector $z \in Z$ is said to *weakly-dominate* an objective vector $z' \in Z$ iff $\forall i \in \{1,\ldots,n\}$, $z_i \leq z'_i$.

Definition 7 (Strict-dominance) An objective vector $z \in Z$ is said to *strictly-dominate* an objective vector $z' \in Z$ iff $\forall i \in \{1,2,\ldots,n\}$, $z_i < z'_i$.

Definition 8 (ε-dominance) Let $\varepsilon > 0$. An objective vector $z \in Z$ is said to *ε-dominate* an objective vector $z' \in Z$ iff $\forall i \in \{1,2,\ldots,n\}$, $(1+\varepsilon) \cdot z_i \leq z'_i$ (Laumanns et al, 2002).

In the frame of a DMLS algorithm, the decision of such a dominance relation will not directly impact the search process, but it will strongly affect the behavior of the main problem-independent search components. In particular, it will be involved during the archiving step, where nondominated solutions are to be computed. Additionally, it can take part in some dominance-based neighborhood exploration strategies, as it will be discussed later in the paper. Of course, different dominance relations can also be defined at distinct steps of the algorithm. However, let us note that the use of weak-dominance into DMLS algorithms may lead to cycling, since solutions mapping to the same objective vector may be accepted (Paquete et al, 2007).

3.2.2 Current Set Selection

The initial phase of a local search iteration deals with the selection of a set of archived solutions whose neighborhood is to be explored. In the first place, note that, if some archive members are marked as visited, they must be discarded from the current set selection for obvious efficiency reasons. Generally speaking, in the frame of the DMLS model presented in the paper, two main classes can be identified:

- An *exhaustive selection*, where the whole set of solutions from the archive is selected.
- A *partial selection*, where only a subset of solutions is selected.

In the first case, every solutions will produce a number of candidate solutions by means of the neighborhood operator (Talbi et al, 2001; Basseur et al, 2002; Angel et al, 2004). In the latter case, existing approaches simply select a number of solutions (usually a single one) at random (Paquete et al, 2004; Laumanns et al, 2004). Up to our knowledge, most DMLS techniques from the literature all fit into these simple approaches, but more sophisticated strategies can be designed. Hence, on the one hand, a subset of well-diversified solutions can be selected with respect to a density measure, as it is the case in many evolutionary multiobjective optimization methods. Popular examples are sharing or crowding (Deb, 2001). For instance, the less crowded solution from the archive can be selected (Aguirre and Tanaka, 2005). Indeed, assuming that solutions close to each other in the decision space are also in the objective space, we can reasonably state that exploring the neighborhood of solutions from an uncrowded region will help producing solutions in this objective-space area. On the other hand, it could turn out to be fruitful to favor the selection of solutions that entered the archive at the previous iteration in order to intensify the search (Knowles and Corne, 2000). On the contrary, visiting solutions in the order in which they were included to the archive would rather diversify the search.

3.2.3 Neighborhood Exploration

From the current set under consideration, a number of candidate solutions must be generated by means of a neighborhood structure. Such a set is obtained by a repeated local transformation of every solution contained in the current set. Similarly to current set selection strategies, for a given solution to be explored, two main classes can be clearly distinguished:

- An *exhaustive neighborhood exploration*, where the neighborhood is evaluated in a complete and deterministic way.
- A *partial neighborhood exploration*, where only a subset of moves are applied.

These two classes can be seen as adaptations of the *best-improving* and the *first-improving* strategies from the single-objective local search literature (Talbi, 2009), respectively. However, for the multiobjective case, multiple improving neighboring solutions can be identified after an exhaustive neighborhood exploration, and different strategies can be designed for a partial neighborhood exploration. These two classes are discussed below.

Exhaustive Neighborhood Exploration. In this case, the complete neighborhood examination of a given solution x is performed. Every possible move is applied and the nondominated neighboring solutions, with respect to x, are all added to the candidate set. Note that, in such an exhaustive neighborhood exploration, solutions from the current set can all be marked as visited at the end of the corresponding step. This gives a strong advantage to this scheme, since the revaluation of some neighbors is avoided, and since a natural stopping condition

can then be met; see Sect. 3.2.5. However, this strategy could appear very time and space consuming in practice, particularly if the size of the neighborhood or of the current set is very large. It may even appear impracticable for some particular applications where the size of the neighborhood is innumerable.

Partial Neighborhood Exploration. In existing DMLS techniques, the number of moves to be applied within a partial neighborhood exploration is generally defined *a priori* by a user-given parameter. For instance, in PAES (Knowles and Corne, 2000), this number is generally set to 1, so that the application of a move is then closely related to a mutation operator from an evolutionary computation point of view. However, a solution neighborhood exploration may continue until a move produces a solution that is better, or not worst, than the current one with respect to a dominance relation. Surprisingly, up to our knowledge, such techniques, though largely employed in single-objective optimization, have not yet been investigated in a DMLS-like algorithm, with the exception of (Aguirre and Tanaka, 2005). But note that equivalent ideas have also been proposed in a repeated single-solution-based multiobjective local search (Ulungu et al, 1999). Let us divide partial neighborhood exploration schemes as follows:

- *Random neighbor.* A single random neighbor *per* current solution is proposed as candidate solution for integrating the archive (Knowles and Corne, 2000).
- *First nondominated neighbor.* For each solution x from the current set, neighbors are evaluated until a nondominated one, with respect to x, is found. This neighbor is then proposed as a candidate for integrating the archive. If no nondominated neighbor is found, the process stops once the neighborhood of x has been examined entirely. In the best case, a single neighbor is to be evaluated, whereas in the worst case (*i.e.* all neighbors of x are dominated by x), the neighborhood is examined completely, so that x can be marked as visited.
- *First dominating neighbor.* For each solution x from the current set, neighbors are evaluated until a dominating one, with respect to x, is found (Aguirre and Tanaka, 2005). Then, all evaluated nondominated neighbors are proposed as candidate solutions for integrating the archive. Similarly to the previous strategy, the possible number of neighbors to be evaluated *per* current solution goes from 1 to $|\mathcal{N}(x)|$.

Of course, these three strategies can easily be extended to the case where more-than-one random, nondominated, or dominating neighbors are to be found. Furthermore, note that a different dominance relation than Pareto-dominance can be employed for a pairwise comparison of solutions; see Sect. 3.2.1. For non-random schemes, note that, in the worst case, a complete evaluation of the neighborhood is performed on each solution from the current set. However, in the case of a *first dominating neighbor* strategy, and given a solution x from the current set, if no improving solution is found, every neighbor is evaluated, so that the current solution can be marked as visited. Thus, assuming that dominated solutions are always discarded from the archive, either the corresponding solution x will be marked as visited, or it will not be included into the archive. As a consequence, the *first dominating neighbor* scheme is the single partial neighborhood exploration strategy that gives rise to a natural stopping condition, and that avoids the revaluation of solutions neighborhood.

3.2.4 Archiving

Another essential issue deals with maintaining the *archive*. This set allows to store either all or a subset of nondominated solutions found during the search process. Its main aim is to prevent the loss of interesting solutions during the stochastic optimization process. But archive

members are also integrated into the search process by providing solutions to exploit in the DMLS model presented in this paper. In the frame of such a model, one may distinguish different archiving techniques depending on the problem properties, the designed algorithm and the number of desired solutions: (*i*) an *unbounded archive* or (*ii*) a *bounded archive*[1]. An archive usually comprises the current nondominated set approximation, as dominated solutions are discarded. Then, an unbounded archive can be used in order to save the whole set of nondominated solutions found until the beginning of the search process. However, as most MCOPs contain an exponential number of nondominated solutions, it becomes impossible to store them all for large-size problem instances. Additionally, it may appear computationally prohibitive to explore the neighborhood of the whole archive exhaustively for some configurations of DMLS algorithms. Therefore, additional operations must be used to reduce the number of solutions to be stored. Then, a common strategy is to bound the size of the archive according to some strategy (Knowles and Corne, 2004). One of the most simple example of bounding technique consists of the following principle (Rudolph and Agapie, 2000). While the archive is below its capacity, all nondominated solutions are incorporated. On the other hand, if the archive content is at its highest level, only dominating solutions are accepted. In both cases, any dominated solution is removed. However, other techniques are based on a diversity criteria. For instance, solutions can be distinguished with respect to a crowding distance (Deb, 2001). The crowding distance can be defined as the circumference of the rectangle defined by a solution's neighbors, with an infinite value for extreme solutions. Otherwise, a relaxed form of dominance can be considered as well. For instance, the ε-dominance concept (Def. 8) can be used to maintain the archive. This allows to reduce the size of the archive without making the process much more complex compared to an unbounded strategy based on the classical Pareto-dominance. However, it is often difficult to set an appropriate ε-value according to the required maximum number of archive items.

3.2.5 Stopping condition

Since an iterative method computes successive approximations, a practical test is generally required to determine when the process must stop. Popular examples are a user-given maximum number of iterations or maximum runtime. However, when it is possible to mark archive members as visited, depending on the neighborhood exploration scheme under consideration (see Sect. 3.2.3), a natural stopping criterion consists of continuing the search process until all archive members are marked as visited. Indeed, in such a case, the archive only contains mutually nondominated solutions whose neighbors are dominated or equivalent to at least one solution from the same archive. As a consequence, the corresponding algorithm falls in a locally efficient set (Paquete et al, 2007). Another strategy is to enumerate the number of consecutive iterations without improvement, and to stop the search process when it exceeds a user-given value. But a non-improving iteration is difficult to define while dealing with multiobjective optimization. For instance, an iteration can be stated as non-improving if the number of new nondominated solutions added to the archive is null.

3.3 State-of-the-art Methods as Instances of the DMLS Model

The aim of this section is to show how state-of-the-art DMLS algorithms conveniently fit into the unified view proposed in the paper. The Pareto Local Search 1 (PLS-1) proposed

[1] As proposed in some multiobjective evolutionary algorithms (Zitzler et al, 2001), a fixed-size archive could be considered as well.

Table 1 State-of-the-art DMLS algorithms as instances of the DMLS model: PLS-1 (Paquete et al, 2004), PLS-2 (Talbi et al, 2001; Basseur et al, 2002; Angel et al, 2004), PAES (Knowles and Corne, 2000), and moRBC$(|A|:1+1)^A$ (Aguirre and Tanaka, 2005).

components	PLS-1	PLS-2	PAES	moRBC$(|A|:1+1)^A$
dominance relation	Pareto dominance	Pareto dominance	Pareto dominance	Pareto dominance
current set selection	*partial* 1 random solution	*exhaustive* all solutions	*partial* μ solutions	*partial* 1 less crowded solution
neighborhood exploration	*exhaustive* all neighbors	*exhaustive* all neighbors	*partial* λ random neighbors	*partial* 1 dominating neighbor
archiving	*unbounded*	*unbounded*	*bounded* adaptive hypergrid	*bounded* crowding distance
stopping condition	*natural* all solutions visited	*natural* all solutions visited	*user-defined*	*natural* all solutions visited

by Paquete et al (2004), the Pareto Local Search 2 (PLS-2) (Talbi et al, 2001; Basseur et al, 2002; Angel et al, 2004), PAES (Knowles and Corne, 2000) and moRBC$(|A|:1+1)^A$ (Aguirre and Tanaka, 2005) are taken as examples. In Table 1, they are treated as simple instances of the DMLS model. Of course, note that only problem-independent components are presented. Firstly, note that all these algorithms handle Pareto-dominance as dominance relation. Thus, at each iteration, PLS-1 selects an unvisited solution at random from the archive, and explores its neighborhood exhaustively. It handles a natural stopping condition that is verified when all archive members are marked as visited. An unbounded archive is generally employed (Paquete et al, 2004), even if a bounding mechanism can be included as well. The only difference between PLS-1 and PLS-2 is that the latter adds the whole set of (unvisited) archive solutions to the current set at each iteration, so that a very large candidate set may potentially be constructed. Next, in PAES, μ solutions are selected from the archive, and a partial neighborhood exploration is performed, with the evaluation of λ random neighbors per solution. In its basic version, note that μ and λ are both set to 1. The PAES current set selection strategy consists in selecting either the current solution from the previous iteration or the latest produced solution, depending on which one is in the less crowded region of the archive. Furthermore, PAES uses a bounded-size archiving technique based on adaptive crowding procedure where the search space is divided by a hypergrid (Knowles and Corne, 2000). The only differences between PAES and the SEMO algorithm from Laumanns et al (2004) is that the latter does not handle any mechanism to bound the size of the archive, and selects the current solution at random. At last, moRBC$(|A|:1+1)^A$ also uses a bounded archive based on crowding distance (Deb, 2001). A single solution, the less crowded one, is then selected from the archive. Next, its neighbors are evaluated until a dominating one, with respect to the current solution, is found, or once the corresponding neighborhood has been completely explored. Every evaluated nondominated neighbor is added to the candidate set. As illustrated in Table 1, these four well-known DMLS-like algorithms are based on simple variations of the problem-independent components presented above.

4 Experimental Analysis

This section provides a comparison of a number of state-of-the-art and original strategies, based on the DMLS model, to solve different kinds of MCOPs. Note that the aim of the experimental part is not to provide the best-performing search method for the particular case of the problems under consideration. Instead, we discuss the respective behaviors of these different strategies, and we try to provide useful guidelines about the choice of the main DMLS-related design issues. This is the reason why we here choose to employ relatively simple and basic problem-related components. Then, current set selection and neighborhood exploration techniques are experimented on both a multiobjective flowshop scheduling problem and a multiobjective traveling salesman problem. These two applications can also be seen as illustrative examples of the DMLS model proposed in the paper. Even if feasible solutions can be represented as permutations for both problems, they differ in the nature of their objective functions, the cardinality of their Pareto fronts, and the solution structure the search operators focus on. The resulting algorithms have all been implemented under ParadisEO 1.2.1 and share the same base components for a fair comparison between them. Moreover, note that the DMLS model has been implemented and integrated into the ParadisEO-MOEO software framework (Liefooghe et al, 2010, 2011). ParadisEO is available for download at the following URL: `http://paradiseo.gforge.inria.fr`.

4.1 Experimental Design

In this section, the strategies under investigation are first presented. Next, we describe the stopping and restarting criteria for all the experimented algorithms as well as the procedure followed for assessing their performance.

4.1.1 Strategies under Investigation

A number of strategies are experimented for current set selection and neighborhood exploration. This gives rise to a combination of 8 DMLS algorithms, as summarized in Fig. 2. Firstly, with regards to current set selection, two simple strategies are investigated, either (*i*) the whole set of solutions or (*ii*) a random single one is selected from the unvisited set of the archive. Next, with regards to neighborhood exploration techniques, three partial strategies and an exhaustive one are studied:

- 1 *random neighbor*. A single neighbor per solution is proposed as a candidate solution for integrating the archive.
- 1 *nondominated neighbor*. For each solution x from the current set, the corresponding first nondominated neighbor, with respect to x, is proposed as a candidate solution for integrating the archive.
- 1 *dominating neighbor*. For each solution x from the current set, neighbors are evaluated until a dominating one, with respect to x, is found. Then, all evaluated nondominated neighbors are proposed as candidate solutions for integrating the archive.
- *All neighbors*. All neighbors of each solution from the current set are proposed as candidate solutions for integrating the archive.

The resulting algorithms are denoted by DMLS $(1 \cdot 1)$, DMLS $(1 \cdot 1_{\not\prec})$, DMLS $(1 \cdot 1_{\succ})$, DMLS $(1 \cdot \star)$, DMLS $(\star \cdot 1)$, DMLS $(\star \cdot 1_{\not\prec})$, DMLS $(\star \cdot 1_{\succ})$, and DMLS $(\star \cdot \star)$, as reported in Fig. 2. Note that some algorithms match, or are closely related to existing approaches from the literature. Indeed, DMLS $(1 \cdot \star)$ is equivalent to PLS-1 (Paquete et al,

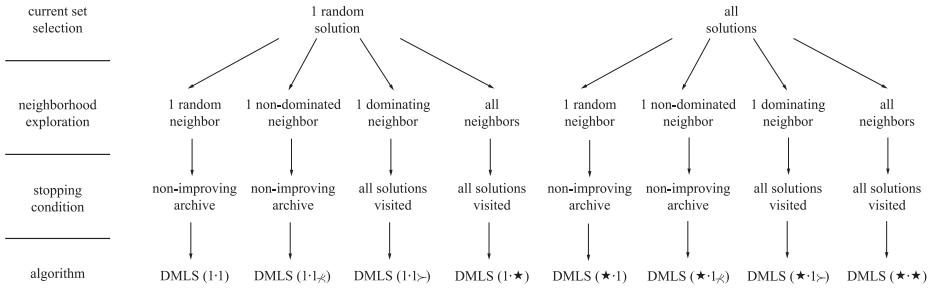

Fig. 2 Algorithms under investigation.

2004), and DMLS $(\star \cdot \star)$ to PLS-2 (Talbi et al, 2001; Basseur et al, 2002; Angel et al, 2004). DMLS $(1 \cdot 1)$ is in somehow related to the SEMO approach of Laumanns et al (2004), and to the (1+1)-PAES approach proposed by Knowles and Corne (2000). However, contrary to the approach of Knowles and Corne (2000), the archiving strategy used here is not based on an adaptive hypergrid. At last, DMLS $(1 \cdot 1_{\succ})$ is closely related to moRBC$(|A| : 1 + 1)^A$ (Aguirre and Tanaka, 2005), except that the latter one uses crowding distance to bound the archive size and to select a solution to explore. The dominance relation used within all the DMLS methods under consideration is Pareto-dominance. The archiving mechanisms used by the DMLS algorithms will be chosen according to the problem to be solved and will be explained in the remaining part of the paper, together with the choice of stopping conditions. Additionally, in order to illustrate the efficiency of such DMLS approaches, we will compare the results obtained by the algorithms of the framework to another state-of-the-art metaheuristic for multiobjective optimization, namely the Non-dominated Sorting Genetic Algorithm II (NSGA-II) proposed by Deb et al (2002).

4.1.2 Stopping and Restarting Conditions

For each problem instance to be solved, a maximum runtime value has been chosen. The current approximation of the efficient set is stored every minute in order to study the evolution of the search efficiency over time. However, we already pointed out that some of the algorithms handle a natural stopping condition. Indeed, when the neighborhood of all archive members has been explored in an exhaustive way, solutions are marked as visited and the algorithm stops; see Sect. 3.2.5. In this case, a simple random restart strategy has been performed to continue the search process until the maximum runtime is reached. Hence, the search is restarted from another random initial population. This mechanism is repeated as many times as necessary to reach the maximum runtime available. However, in order not to penalize the algorithms that do not stop naturally, a simple non-improving stopping condition has been designed. The latter is verified when a number of consecutive iterations are performed without any new nondominated solution incorporating the archive. This parameter is set to v for exhaustive selection strategies, and to $(v \times |A|)$ for single solution-based selection strategies, where v stands for the number of neighbors per solution and $|A|$ for the size of the current archive. If the current iteration is stated as non-improving, note that $|A|$ stays constant over time. This parameter values are motivated by the fact that, with such a number of iterations, the corresponding algorithms would have had the time to generate the complete neighborhood of the actual nondominated set. The stopping condition corresponding to any algorithm under investigation is reported in Fig. 2.

4.1.3 Performance Assessment

In the frame of multiobjective optimization, the performance assessment of a number of algorithms in solving the same problem is a key issue. In this study, a set of 20 runs *per* instance and *per* algorithm is performed. In order to evaluate the quality of the nondominated front approximations for every instance we experimented, we follow the protocol proposed in (Knowles et al, 2006). For a given instance, let Z^{all} denote the union of the outputs we obtained during all our experiments. Note that Z^{all} probably contains both dominated and nondominated vectors, as a given approximation may contain vectors dominating the ones of another approximation, and *vice versa*. We first compute a reference set $Z_N^\star$ containing all the nondominated vectors of Z^{all} plus any other existing best know nondominated set for the problem under consideration. Second, we define $z^{min} = (z_1^{min}, \ldots, z_n^{min})$ and $z^{max} = (z_1^{max}, \ldots, z_n^{max})$, where z_k^{min} (resp. z_k^{max}) denotes the lower (resp. upper) bound of the k^{th} objective for all the points contained in Z^{all}. In order to give a roughly equal range to the objective functions, values are normalized with respect to z^{min} and z^{max}.

Let us consider an efficient set approximation A. In order to measure the quality of A in comparison to $Z_N^\star$, we compute the difference between these two sets by using the unary hypervolume metric (Zitzler et al, 2003), z^{max} being the reference point. The hypervolume difference indicator (I_H^-) computes the portion of the objective space that is weakly-dominated by $Z_N^\star$ and not by A. The closer this measure to *zero*, the better the approximation A. Furthermore, we also consider the additive ε-indicator (Zitzler et al, 2003). The unary additive ε-indicator ($\mathrm{I}_{\varepsilon+}^1$) gives the minimum factor by which an approximation A can or has to be translated in the objective space to weakly-dominate the reference set $Z_N^\star$.

As a consequence, for each test instance, we obtain 20 I_H^--values and 20 $\mathrm{I}_{\varepsilon+}^1$-values, corresponding to the 20 runs, *per* algorithm. Once all these values are computed, we first report an average value *per* metric in order to study the evolution of all methods over time. Additionally, we perform a statistical analysis for a pairwise comparison of methods. To this end, we use the Wilcoxon signed rank test. Such a non-parametric statistical test is motivated by the fact that samples collected here correspond to matched samples. Details for this statistical testing procedure are given by Knowles et al (2006). Hence, for a given test instance, and with respect to the indicator under consideration, this statistical test reveals if the sample of approximation sets obtained by a given search method is significantly better than the one of another search method, or if there is no significant difference between both. For the sake of conciseness, we only report how many algorithms obtained statistically better results than the corresponding algorithm for the instance under consideration. In other words, a value of *zero* means that no other method generated significantly better approximations. Note that all the performance assessment procedures have been achieved using the performance assessment tools provided in PISA (Bleuler et al, 2003). The package is available at the URL: `http://www.tik.ee.ethz.ch/pisa/assessment.php`.

4.2 Application 1: Permutation Flowshop Scheduling Problem

The Flowshop Scheduling Problem (FSP) is one of the most investigated scheduling problems of the literature. Most works consider it on a single-objective form and mainly aims at minimizing the *makespan*, *i.e.* the total completion time. However, many objective functions can be taken into account, and numerous multiobjective approaches have also been proposed. The reader is referred to (T'Kindt and Billaut, 2002; Landa Silva et al, 2004) for a survey on multiobjective scheduling. Note that both stand-alone and hybrid DMLS-like

algorithms have already been applied with success to biobjective flowshop scheduling problems (Basseur et al, 2002; Geiger, 2007; Dubois-Lacoste et al, 2009). Here, some results are also reported for the three-objective case.

4.2.1 Problem Description

The FSP consists of scheduling N jobs $\{J_1, J_2, \ldots, J_N\}$ on M machines $\{M_1, M_2, \ldots, M_M\}$. Machines are critical resources, *i.e.* two jobs cannot be assigned to the same machine at the same time. A job J_i is composed of M tasks $\{t_{i1}, t_{i2}, \ldots, t_{iM}\}$, where t_{ij} is the j^{th} task of J_i, requiring the machine M_j. A processing time p_{ij} is associated with each task t_{ij}, and a due date d_i is given to every job J_i. We here focus on a permutation FSP, where the operating sequences of the jobs are identical and unidirectional for every machine. In this study, we will consider a two-objective FSP (denoted by FSP-2), where both the makespan (C_{max}) and the total tardiness ($\overline{T}$) are to be minimized. Additionally, we will also consider a three-objective variant (denoted by FSP-3), where the maximum tardiness (T_{max}) is the additional objective to be minimized. For each task t_{ij} being scheduled at time s_{ij}, these objective functions, that are among the most widely investigated ones of the literature (T'Kindt and Billaut, 2002), can be computed as follows:

$$C_{max} = \max_{i \in \{1,\ldots,N\}} \{s_{iM} + p_{iM}\} \tag{2}$$

$$\overline{T} = \sum_{i=1}^{N} \Big\{ \max\{0, s_{iM} + p_{iM} - d_i\} \Big\} \tag{3}$$

$$T_{max} = \max_{i \in \{1,\ldots,N\}} \Big\{ \max\{0, s_{iM} + p_{iM} - d_i\} \Big\} \tag{4}$$

According to Graham et al (1979), FSP-2 can be denoted by $F/perm, d_i/(C_{max}, \overline{T})$, and FSP-3 by $F/perm, d_i/(C_{max}, \overline{T}, T_{max})$. However, since minimizing the makespan, the total tardiness or the maximum tardiness independently is already known to be strongly NP-hard for more than two machines (T'Kindt and Billaut, 2002), so are the FSP-2 and the FSP-3. As a consequence, large-size problem instances can generally not be solved by exact methods.

4.2.2 Problem-related Components

The problem-related components used for the specific case of the FSP presented above are the following ones:

- *Representation.* Ordered sequence of jobs to be scheduled. A feasible solution for a problem instance of N jobs and M machines is represented by a permutation of size N.
- *Initialization.* Randomly generated solution.
- *Evaluation.* Makespan and total tardiness for FSP-2; makespan, total tardiness and maximum tardiness for FSP-3.
- *Neighborhood. Insertion* operator, *i.e.* a job located at position i is inserted at position $j \neq i$. The jobs located between positions i and j are shifted, as illustrated in Fig. 3. The number of neighbors per solution is $(N-1)^2$, where N stands for the size of a permutation. Note that no particular order is considered to explore the neighborhood of a given solution; neighbors are examined in a random order.
- *Incremental evaluation.* None, each neighboring solution is evaluated from scratch.

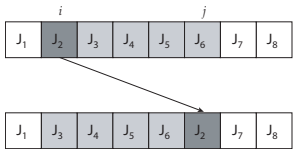

Fig. 3 Illustration of the *insertion* neighborhood operator for the FSP.

4.2.3 Benchmark Test Instances

To evaluate the performance of the algorithms, we consider a set of benchmark test instances (Liefooghe et al, 2007). They have been built from the single-objective instances from Taillard (1993) and extended to the multiobjective case by adding a due date for every job. These instances are available at the URL: `http://www.lifl.fr/~liefooga/benchmarks/`. As proposed by Taillard (1993), processing times are first randomly generated with respect to a uniform distribution in the interval $[0,99]$. Next, we compute the average value of those processing times. Let $\overline{p}$ denote the average processing time value for the instance under consideration. Due dates are then unifromly generated using a random value between $\overline{p} \times M$ and $\overline{p} \times (N+M-1)$, where N stands for the number of jobs and M for the number of machines. Thus, the due date of a given job roughly lies between the average completion date of the first job from the schedule and the average completion date of the last job from the schedule. An instance denoted by $N \times M \times i$ refers to the i^{th} instance with N jobs and M machines. In this paper, we experiment a set of 8 problem instances involving from 20 jobs and 5 machines to 100 jobs and 20 machines.

4.2.4 Parameter Setting

For all the experiments, the initial population size is set to 1. The stopping condition is fixed with respect to the size of the instance under consideration. Hence, we set a maximum runtime of: 10 minutes for 20-job instances, 20 minutes for 50-job instances, and 30 minutes for 100-job instances. At last, due to the reasonable number of nondominated solutions found during our experiments, an unbounded archive is here maintained. The parameters used for NSGA-II are the following ones: a population size of 100 solutions, a crossover rate of 0.8 and a mutation rate of 1.0. Note that the crossover operator used for the FSP is the two-point crossover proposed by Ishibuchi and Murata (1998), and the mutation operator is based on the insertion neighborhood operator used for the DMLS.

4.2.5 Experimental Results and Discussion

The set of FSP-related experiments has been conducted on an Intel Core 2 Duo 6600 ($2 \times$ 2.40 GHz, 2 GB RAM) machine, with GCC 4.1.2 running under Linux. FSP-2 and FSP-3 results are presented in Table 2 for the I_H^- indicator. Due to space limitations, we highlight only the statistical outputs for three different stopping criteria, fixed according to the instance under consideration: a small, a medium and a large runtime. The table for the $\mathrm{I}_{\varepsilon+}^1$ indicator as well as visual plots of the average I_H^-- and $\mathrm{I}_{\varepsilon+}^1$-values over time are available at the URL: `http://www.lifl.fr/~liefooga/sup/dmls/`. First of all, let us remark that two distinct sets of instances can clearly be distinguished with respect to algorithmic performance. The first set contains relatively small-size instances, including 20-job and 50-job

Table 2 Comparison of DMLS algorithms with respect to the I_H^- indicator for FSP benchmark test instances. The first value stands for the number of algorithms that statistically outperform the one under consideration. The number in brackets stands for the normalized average I_H^--value ($\times 10^{-1}$).

Instance	Runtime	DMLS $(1\cdot 1)$	DMLS $(1\cdot 1_{\not\succ})$	DMLS $(1\cdot 1_{\succ})$	DMLS $(1\cdot \star)$	DMLS $(\star\cdot 1)$	DMLS $(\star\cdot 1_{\not\succ})$	DMLS $(\star\cdot 1_{\succ})$	DMLS $(\star\cdot\star)$	NSGA-II
FSP-2										
$020\times 05\times 01$	1'	4 (0.511)	6 (1.378)	0 (0.034)	1 (0.055)	5 (0.882)	5 (1.146)	0 (0.021)	3 (0.133)	8 (2.934)
	5'	4 (0.055)	6 (0.663)	0 (0.003)	1 (0.005)	6 (0.362)	4 (0.080)	0 (0.004)	1 (0.005)	8 (2.856)
	10'	4 (0.027)	6 (0.199)	0 (0.002)	2 (0.003)	7 (0.189)	4 (0.036)	0 (0.002)	3 (0.004)	8 (2.841)
$020\times 10\times 01$	1'	4 (0.852)	4 (0.847)	1 (0.385)	1 (0.373)	7 (1.886)	4 (0.813)	0 (0.235)	1 (0.420)	7 (2.326)
	5'	4 (0.327)	4 (0.372)	0 (0.063)	2 (0.181)	7 (0.865)	4 (0.376)	0 (0.074)	0 (0.117)	8 (1.812)
	10'	4 (0.178)	5 (0.267)	0 (0.021)	2 (0.080)	7 (0.524)	4 (0.233)	0 (0.020)	2 (0.081)	8 (1.608)
$020\times 20\times 01$	1'	3 (1.490)	4 (1.732)	0 (0.463)	2 (0.897)	7 (2.894)	4 (1.530)	1 (0.623)	2 (1.143)	8 (5.549)
	5'	4 (0.466)	6 (0.858)	0 (0.136)	1 (0.200)	7 (1.514)	4 (0.539)	0 (0.175)	3 (0.295)	8 (4.799)
	10'	4 (0.243)	6 (0.611)	0 (0.052)	2 (0.142)	7 (1.042)	5 (0.332)	1 (0.077)	2 (0.149)	8 (4.668)
$050\times 05\times 01$	2'	3 (1.114)	2 (1.123)	0 (0.793)	2 (0.987)	4 (1.216)	2 (1.047)	0 (0.771)	7 (1.812)	8 (2.658)
	10'	3 (0.737)	6 (0.880)	0 (0.475)	1 (0.545)	5 (0.876)	4 (0.795)	0 (0.396)	2 (0.665)	8 (2.466)
	20'	3 (0.592)	6 (0.792)	0 (0.289)	1 (0.359)	5 (0.756)	5 (0.728)	0 (0.262)	3 (0.504)	8 (2.461)
$050\times 10\times 01$	2'	1 (1.519)	0 (1.443)	0 (1.372)	5 (1.849)	5 (1.707)	0 (1.406)	0 (1.337)	8 (5.156)	7 (3.027)
	10'	3 (1.070)	5 (1.296)	0 (0.603)	2 (0.927)	3 (1.157)	5 (1.316)	0 (0.652)	2 (1.062)	8 (2.514)
	20'	4 (0.876)	6 (1.271)	0 (0.467)	2 (0.611)	5 (1.014)	5 (1.187)	0 (0.508)	3 (0.758)	8 (2.440)
$050\times 20\times 01$	2'	0 (1.225)	0 (1.274)	0 (1.302)	5 (1.794)	3 (1.413)	0 (1.220)	5 (1.698)	8 (5.864)	7 (2.202)
	10'	1 (0.783)	5 (1.069)	0 (0.666)	1 (0.781)	4 (0.909)	3 (0.964)	0 (0.732)	7 (1.644)	7 (1.737)
	20'	2 (0.652)	6 (1.060)	0 (0.477)	0 (0.564)	4 (0.810)	5 (0.937)	0 (0.476)	2 (0.739)	8 (1.663)
$100\times 10\times 01$	3'	4 (1.164)	0 (0.601)	1 (0.790)	7 (3.638)	4 (1.125)	0 (0.700)	2 (0.874)	8 (7.961)	6 (2.051)
	15'	1 (0.484)	0 (0.285)	1 (0.431)	6 (1.277)	1 (0.463)	1 (0.410)	1 (0.448)	8 (5.820)	7 (1.648)
	30'	2 (0.354)	0 (0.213)	1 (0.300)	6 (0.741)	1 (0.296)	1 (0.348)	1 (0.325)	8 (4.540)	7 (1.565)
$100\times 20\times 01$	3'	2 (1.754)	0 (1.335)	2 (1.720)	7 (5.637)	2 (1.703)	0 (1.334)	4 (1.928)	8 (8.902)	6 (2.432)
	15'	0 (0.824)	0 (0.734)	4 (1.062)	7 (2.332)	0 (0.692)	0 (0.735)	4 (1.144)	8 (7.426)	6 (1.658)
	30'	0 (0.544)	0 (0.513)	4 (0.759)	6 (1.483)	0 (0.485)	0 (0.577)	4 (0.845)	8 (6.354)	6 (1.489)
FSP-3										
$020\times 05\times 01$	1'	5 (0.928)	3 (0.524)	0 (0.231)	2 (0.400)	5 (1.057)	2 (0.413)	0 (0.214)	7 (1.313)	8 (4.653)
	5'	4 (0.319)	4 (0.309)	0 (0.040)	2 (0.087)	4 (0.318)	2 (0.070)	0 (0.034)	4 (0.250)	8 (3.846)
	10'	5 (0.180)	5 (0.208)	1 (0.023)	2 (0.045)	5 (0.197)	2 (0.035)	0 (0.017)	3 (0.104)	8 (3.677)
$020\times 10\times 01$	1'	4 (1.184)	4 (0.992)	0 (0.386)	2 (0.698)	6 (1.782)	2 (0.880)	0 (0.451)	6 (2.505)	8 (3.829)
	5'	4 (0.417)	5 (0.524)	0 (0.116)	2 (0.234)	7 (0.685)	3 (0.407)	0 (0.083)	3 (0.344)	8 (2.301)
	10'	5 (0.276)	6 (0.367)	0 (0.043)	2 (0.128)	6 (0.436)	4 (0.224)	0 (0.039)	2 (0.152)	8 (1.996)
$020\times 20\times 01$	1'	2 (1.714)	1 (1.451)	0 (1.041)	3 (1.682)	6 (2.066)	2 (1.626)	0 (1.118)	7 (4.945)	7 (5.002)
	5'	3 (0.622)	5 (0.890)	0 (0.178)	2 (0.319)	5 (0.907)	5 (0.952)	0 (0.180)	3 (0.620)	8 (3.541)
	10'	4 (0.375)	5 (0.522)	0 (0.097)	2 (0.159)	6 (0.600)	5 (0.464)	0 (0.069)	3 (0.320)	8 (3.228)
$050\times 05\times 01$	2'	4 (0.864)	0 (0.428)	2 (0.725)	6 (1.640)	2 (0.673)	0 (0.400)	3 (0.815)	8 (8.373)	6 (1.693)
	10'	3 (0.377)	1 (0.332)	0 (0.266)	6 (0.469)	3 (0.339)	0 (0.291)	0 (0.292)	8 (6.719)	7 (1.354)
	20'	2 (0.308)	2 (0.314)	0 (0.175)	2 (0.284)	2 (0.297)	2 (0.283)	1 (0.226)	8 (5.412)	7 (1.300)
$050\times 10\times 01$	2'	3 (1.546)	0 (1.087)	4 (1.836)	6 (2.780)	2 (1.358)	0 (1.058)	5 (2.495)	8 (8.698)	6 (2.917)
	10'	3 (0.662)	0 (0.499)	4 (0.856)	5 (1.439)	1 (0.558)	0 (0.477)	5 (1.406)	8 (7.400)	7 (2.419)
	20'	2 (0.436)	0 (0.335)	2 (0.454)	5 (0.947)	0 (0.368)	0 (0.320)	5 (0.962)	8 (6.522)	7 (2.233)
$050\times 20\times 01$	2'	3 (1.761)	1 (1.509)	4 (2.489)	5 (3.385)	1 (1.582)	0 (1.330)	6 (3.392)	8 (8.961)	5 (3.009)
	10'	1 (0.751)	2 (0.808)	4 (1.175)	5 (1.626)	0 (0.664)	0 (0.672)	6 (2.063)	8 (7.889)	6 (2.276)
	20'	0 (0.500)	3 (0.605)	2 (0.602)	5 (1.106)	0 (0.412)	0 (0.451)	6 (1.470)	8 (7.094)	7 (2.090)
$100\times 10\times 01$	3'	4 (3.115)	0 (2.102)	1 (2.171)	7 (6.450)	3 (2.872)	0 (2.012)	6 (3.559)	8 (9.466)	4 (3.155)
	15'	3 (1.192)	0 (0.554)	2 (1.038)	7 (3.874)	3 (1.213)	0 (0.577)	5 (1.728)	8 (9.240)	6 (2.253)
	30'	2 (0.716)	0 (0.228)	2 (0.772)	7 (2.981)	2 (0.700)	0 (0.235)	5 (1.186)	8 (9.084)	6 (2.026)
$100\times 20\times 01$	3'	2 (4.036)	0 (3.354)	2 (3.908)	7 (8.127)	2 (3.843)	1 (3.593)	6 (5.868)	8 (9.743)	5 (4.398)
	15'	2 (1.797)	0 (1.063)	4 (2.117)	7 (5.193)	2 (1.676)	1 (1.370)	6 (3.919)	8 (9.540)	5 (3.302)
	30'	3 (1.084)	0 (0.412)	4 (1.590)	7 (3.888)	2 (0.973)	1 (0.594)	5 (3.054)	8 (9.441)	5 (2.983)

FSP-2 instances as well as 20-job FSP-3-instances. The second set contains large-size instances and comprises 50-job FSP-3 instances as well as 100-job FSP-2 and FSP-3 instances. In terms of size, we then speak here both about the size of the search space and the number of objective functions.

For the small-size set, the two best-performing algorithms are clearly DMLS $(1\cdot 1_{\succ})$ and DMLS $(\star\cdot 1_{\succ})$. The only exceptions are the $020\times 10\times 01$ and the $050\times 20\times 01$ FSP-2 instances, where DMLS $(1\cdot 1_{\succ})$ (resp. DMLS $(\star\cdot 1_{\succ})$) is outperformed by at least one other algorithm, including DMLS $(\star\cdot 1_{\succ})$ (resp. DMLS $(1\cdot 1_{\succ})$), when a short amount of runtime is allowed. However, DMLS $(1\cdot\star)$ seems to be a solid competitor. As well, the results obtained by DMLS $(\star\cdot\star)$ are not too bad for very small FSP-2 instances, but the latter can generally not compete with others for larger instances. At last, except for a few exceptions, all the other methods, including NSGA-II, globally obtain very poor performance in comparison to the methods discussed above. Thus, we can reasonably conclude that stopping the neighborhood exploration once an improving neighbor, in terms of Pareto-dominance, is found for each current solution, is the best strategy to adopt for this set of instances, whatever the current set selection scheme under consideration.

Table 3 Average number of random restart operations for FSP benchmark test instances. The size of the best nondominated set for the instance under consideration is given as information in the second column ($|Z_N^\star|$).

Instance	$\lvert Z_N^\star\rvert$	DMLS $(1\cdot 1)$	DMLS $(1\cdot 1_{\not\succ})$	DMLS $(1\cdot 1_{\succ})$	DMLS $(1\cdot\star)$	DMLS $(\star\cdot 1)$	DMLS $(\star\cdot 1_{\not\succ})$	DMLS $(\star\cdot 1_{\succ})$	DMLS $(\star\cdot\star)$
FSP-2									
$020\times 05\times 01$	18	977.80	86.15	2878.20	1100.05	4932.15	206.35	2228.25	430.70
$020\times 10\times 01$	24	458.85	47.10	392.10	226.25	1789.10	68.55	265.05	97.75
$020\times 20\times 01$	30	329.30	18.85	220.10	151.95	1239.65	32.30	145.70	66.85
$050\times 05\times 01$	28	85.20	4.70	77.40	22.05	141.15	4.50	56.40	6.40
$050\times 10\times 01$	48	37.10	1.20	14.45	7.35	63.75	0.30	8.85	1.60
$050\times 20\times 01$	38	18.70	1.00	6.40	4.00	46.75	0.00	3.45	0.70
$100\times 10\times 01$	89	1.00	0.00	0.40	0.00	0.00	0.00	0.00	0.00
$100\times 20\times 01$	76	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
FSP-3									
$020\times 05\times 01$	278	19.85	11.65	52.00	17.40	273.15	35.20	34.60	6.60
$020\times 10\times 01$	191	38.40	10.65	75.00	35.05	406.80	23.15	42.80	10.95
$020\times 20\times 01$	423	11.85	3.10	18.40	9.80	133.10	2.00	8.75	3.15
$050\times 05\times 01$	273	1.65	1.00	3.70	0.70	0.00	0.00	1.70	0.00
$050\times 10\times 01$	969	1.10	1.00	0.00	0.00	0.00	0.00	0.00	0.00
$050\times 20\times 01$	592	1.25	1.00	0.00	0.00	0.00	0.00	0.00	0.00
$100\times 10\times 01$	696	1.00	1.00	0.00	0.00	0.00	0.00	0.00	0.00
$100\times 20\times 01$	618	1.00	1.00	0.00	0.00	0.00	0.00	0.00	0.00

For large-size instances, the more efficient neighborhood exploration strategy clearly corresponds to the "first nondominated neighbor" scheme. Indeed, both DMLS $(1\cdot 1_{\not\succ})$ and DMLS $(\star\cdot 1_{\not\succ})$ are never dominated by another method at the same time. However, even if it is often outperformed by methods mentioned above, DMLS $(1\cdot 1_{\succ})$ obtain good performance also on large-size instances on average, contrary to DMLS $(\star\cdot 1_{\succ})$ that presents poor performance, especially for FSP-3 instances. Besides, the results obtained by DMLS $(\star\cdot\star)$ are very poor, corroborating the results highlighted from the first set of instances. So are the ones of DMLS $(1\cdot\star)$, in opposite to small-size instances. On the contrary, quite simple methods like DMLS $(1\cdot 1)$ and DMLS $(\star\cdot 1)$ obtain good performance on large instances. At last, the performances of NSGA-II seem to be lithly higher than for small-size problems.

To summarize, there does not exist a unambiguous strategy to adopt in terms of current set selection. However, stopping the exploration of a solution neighborhood once the first dominating neighbor is found appears to be very efficient for small instances, whereas stopping it once the first nondominated neighbor is found performs better an larger problems. Such a behavior could be related to the number of restarts performed by the corresponding algorithms. Indeed, as pointed out in Table 3, both DMLS $(1\cdot 1_{\succ})$ and DMLS $(\star\cdot 1_{\succ})$ restart more often than DMLS $(1\cdot 1_{\not\succ})$ and DMLS $(\star\cdot 1_{\not\succ})$ for small instances, where they perform better. But for large instances, where the latter methods outperform the former ones, the number of restarts is close to zero and then more or less comparable between both. Note that this might also be related to the number of points located in the trade-off surface for the instance under consideration. As reported in Table 3, this number gets bigger for large-size instances. As a matter of fact, we can reasonably assume that, the larger the number of efficient solutions, the larger the number of nondominated points are actually found by the algorithms, and then the larger the size of the archive is. Consequently, the chance to fall in a kind of Pareto local optimum set is reduced, and so is the chance to restart for all methods.

4.3 Application 2: Traveling Salesman Problem

The Traveling Salesman Problem (TSP) is probably the most often investigated combinatorial optimization problem. In its usual single-objective form, the TSP aims at finding a tour

such as a single total cost has to be minimized. However, in practice, additional costs such as distance or travel time are to be considered as well. This may explain the reason why the TSP started to be widely investigated in a multiobjective way in recent years; see for instance (Jaszkiewicz, 2002; Paquete and Stützle, 2003; Angel et al, 2004). Successful applications of DMLS algorithms for the biobjective TSP can be found in (Angel et al, 2004; Paquete and Stützle, 2009; Lust and Teghem, 2010; Lust and Jaszkiewicz, 2010). Note that Paquete and Stützle (2009) also performed a deep analysis of problem-related components (initialization, neighborhood) involved in a DMLS algorithm for the biobjective TSP.

4.3.1 Problem Description

The TSP can be defined by a complete graph $G = (V,E)$ where $V = \{v_1, v_2, \ldots, v_p\}$ is a set of nodes and $E = \{[v_i, v_j] | v_i, v_j \in V\}$ is a set of edges. In the n-objective case, we have n cost matrices C^k, $k = 1, ..., n$. To each edge $[v_i, v_j] \in E$ is then assigned a non-negative *cost* c_{ij}^k for each objective function $k \in \{1, \ldots, n\}$. The aim is to find simple cyclic permutations π of $\{v_1, \ldots, v_p\}$ (with $\pi(p) = v_1$) that minimize $(f_1(\pi), \ldots, f_n(\pi))$, such that:

$$f_k(\pi) = c^k_{\pi(p)\pi(1)} + \sum_{i=1}^{n-1} c^k_{\pi(i)\pi(i+1)} \tag{5}$$

The multiobjective TSP is NP-hard (Serafini, 1986). TSP variants with two and three objective functions are considered here. They will be respectively denoted by TSP-2 and TSP-3.

4.3.2 Problem-related Components

The TSP-specific components used for the design of local search methods are:

- *Representation.* Ordered sequence of nodes, excluding v_1. A feasible solution for a problem instance of p nodes is represented by a permutation of size $(p-1)$.
- *Evaluation:* Two cost matrices are considered for TSP-2, and three ones for TSP-3.
- *Initialization.* Randomly generated solution.
- *Neighborhood. 2-opt exchange* operator, *i.e.* the sequence located between $\pi(i)$ and $\pi(j)$ is reversed. Hence, there exists a set of $\frac{(p-1)\cdot(p-2)}{2}$ neighbors per solution.
- *Incremental evaluation.* For every objective function f_k with $k \in \{1, \ldots, n\}$, subtracting $(c^k_{\pi(i-1)\pi(i)} + c^k_{\pi(j)\pi(j+1)})$ and adding $(c^k_{\pi(i-1)\pi(j)} + c^k_{\pi(i)\pi(j+1)})$. Neighbors are examined in a random order.

4.3.3 Benchmark Test Instances

Each objective function of a multiobjective TSP is computed with an independent cost matrix, so that a number of 2 (*resp.* 3) cost matrices is required for TSP-2 (*resp.* TSP-3). In the literature, the set of Krolak, Fets and Nelson euclidean instances available in the TSPLIB (files prefixed by *Kro*) are in general tackled for the multiobjective TSP (Jaszkiewicz, 2002; Paquete and Stützle, 2003; Paquete et al, 2004; Angel et al, 2004). However, the size of these instances is equal or below 200 nodes. This is the reason why Paquete and Stützle (2009) proposed euclidean costs matrices built in the same way, but involving a larger number of nodes, *i.e.* 100, 300 and 500 nodes. These data allow to build a set of 9 TSP-2 and 6 TSP-3 euclidean instances that are used in our experiments. The authors also make their best found nondominated front available on the web, so that a good reference set exists for those instances. TSP instances as well as their corresponding best found nondominated set are available at the URL: `http://eden.dei.uc.pt/~paquete/tsp/`.

4.3.4 Parameter Setting

Similarly to what has been done for the FSP, the initial population size is set to 1, and the stopping condition is fixed with respect to the size of the instance under consideration. The maximum runtime is set as follows: 10 minutes for 100-node instances, 20 minutes for 300-node instances, and 30 minutes for 500-node instances. Due to the large number of nondominated solutions generally found for both TSP-2 and TSP-3, a bounding archiving mechanism is here required. However, since it will always rely on an assumed trade-off between efficiency and effectiveness, the choice of a proper strategy is subject to endless discussion and is closely related to the problem property and the maximum runtime available. Thus, we here choose to employ a very simple, but computationally effective, bounding mechanism as described in Sect. 3.2.4. A new nondominated solution x is added to the archive only if (*i*) the archive is not full or (*ii*) x dominates at least one solution from the archive, so that one of the dominated solutions is replaced by the new one. All dominated solutions are then discarded. In addition to its obvious simplicity, this strategy has the advantage of being reasonable in terms of time complexity. The maximum archive size is set to 100 for all instances. The NSGA-II parameters are identical to the ones used for the FSP. The recombination operator consists in an order crossover (Davis, 1985).

4.3.5 Experimental Results and Discussion

All the TSP-related experiments have been conducted on an Intel Core 2 Quad 6600 ($4 \times$ 2.40 GHz, 3 GB RAM) machine, with GCC 4.1.3 running under Linux. Results are reported in Table 4 for the I_H^- indicator. As for FSP experiments, note that the counterpart for the $\mathrm{I}_{\varepsilon+}^1$ indicator, together with the plots of the average I_H^-- and $\mathrm{I}_{\varepsilon+}^1$-values over time for the TSP are available at the URL: `http://www.lifl.fr/~liefooga/sup/dmls/`.

The statistical tests performed for both TSP-2 and TSP-3 clearly indicate the high performance of DMLS $(1 \cdot 1_{\succ})$ for most instances. Indeed, this method is never statistically outperformed for 300- and 500-node TSP-2 instances as well as the whole set of TSP-3 instances. However, note that DMLS $(\star \cdot 1_{\succ})$ almost always come at second position, except for 100-node instances. Now, for 100-node TSP-2 instances, DMLS $(1 \cdot 1_{\not\prec})$ and DMLS $(\star \cdot 1_{\not\prec})$ globally seem to be more suited. These two methods also obtain quite satisfying results for larger instances, and for 100-node TSP-3 instances as well, at least once a large amount of computational time is allowed. NSGA-II and the remaining DMLS methods investigated in this paper generally obtain very poor results when compared to the ones mentioned above. NSGA-II becomes also competitive for large-size problem instances, in terms of search space and number of objective functions. Some DMLS algorithms are, however, known to be very efficient, especially the DMLS $(1 \cdot \star)$ algorithm that corresponds to PLS-1. Indeed, PLS-1 is reported to be one of the best performing search method with regards to biobjective TSP solving in the specialized literature (Paquete et al, 2004). But note that the approach investigated by the authors reports better results when using the 3-opt exchange neighborhood operator. Hence, we believe that measuring the efficiency of DMLS $(1 \cdot 1_{\succ})$ within such an operator could be attractively investigated in future research. Finally, let us note that, in comparison to the FSP, experiments performed on the TSP does not lead to a large number of restart operations for most algorithms and instances. Indeed, as reported in Table 5, all methods never restart for both TSP-2 and TSP-3 instances involving 300 nodes and more. Only 100-node problems give rise to several restarts for some approaches. This seems to be related to the high number of non-dominated points generally obtained for TSP instances, in contrast to the FSP.

Table 4 Comparison of DMLS algorithms with respect to the I_H^- indicator for TSP benchmark test instances. The first value stands for the number of algorithms that statistically outperform the one under consideration. The number in brackets stands for the normalized average I_H^--value ($\times 10^{-1}$). Some results for the DMLS $(\star \cdot \star)$ strategy are not available because of memory issues due to the large size of the current set.

Instance	Runtime	DMLS $(1 \cdot 1)$	DMLS $(1 \cdot 1_{\not\prec})$	DMLS $(1 \cdot 1_{\succ})$	DMLS $(1 \cdot \star)$	DMLS $(\star \cdot 1)$	DMLS $(\star \cdot 1_{\not\prec})$	DMLS $(\star \cdot 1_{\succ})$	DMLS $(\star \cdot \star)$	NSGA-II
TSP-2										
*euclidAB*100	1'	6 (2.769)	3 (1.166)	0 (0.828)	4 (2.274)	5 (2.644)	2 (1.132)	1 (0.973)	8 (9.379)	7 (3.782)
	5'	5 (0.959)	0 (0.200)	2 (0.327)	5 (0.943)	4 (0.889)	0 (0.195)	2 (0.274)	8 (8.400)	7 (2.173)
	10'	5 (0.479)	0 (0.120)	3 (0.230)	6 (0.650)	4 (0.445)	0 (0.106)	2 (0.176)	8 (7.329)	7 (1.735)
*euclidCD*100	1'	5 (2.815)	2 (1.201)	0 (0.907)	4 (2.367)	5 (2.784)	2 (1.164)	1 (1.056)	8 (9.535)	7 (4.073)
	5'	5 (0.998)	0 (0.251)	3 (0.470)	6 (1.055)	4 (0.913)	0 (0.223)	2 (0.353)	8 (8.524)	7 (2.286)
	10'	4 (0.487)	0 (0.154)	3 (0.348)	6 (0.825)	4 (0.451)	0 (0.131)	2 (0.258)	8 (7.370)	7 (1.807)
*euclidEF*100	1'	6 (2.868)	3 (1.213)	0 (0.812)	4 (2.455)	5 (2.798)	2 (1.139)	1 (1.015)	8 (9.652)	7 (4.098)
	5'	5 (1.010)	0 (0.224)	3 (0.419)	5 (1.028)	4 (0.902)	0 (0.218)	2 (0.306)	8 (8.658)	7 (2.295)
	10'	5 (0.501)	0 (0.121)	3 (0.251)	6 (0.728)	4 (0.448)	0 (0.129)	2 (0.178)	8 (7.533)	7 (1.805)
*euclidAB*300	2'	5 (8.152)	3 (7.043)	0 (1.952)	4 (8.029)	7 (8.530)	4 (7.786)	1 (5.680)	-	2 (6.378)
	10'	6 (5.932)	2 (3.942)	0 (0.357)	3 (4.330)	7 (6.188)	3 (4.542)	1 (0.742)	-	5 (4.779)
	20'	6 (4.857)	2 (2.479)	0 (0.177)	4 (3.222)	7 (5.033)	3 (2.871)	1 (0.243)	-	5 (4.111)
*euclidCD*300	2'	5 (7.878)	3 (6.935)	0 (1.879)	4 (7.717)	7 (8.290)	4 (7.631)	1 (5.570)	-	2 (6.010)
	10'	6 (5.812)	2 (3.921)	0 (0.361)	3 (4.343)	7 (6.067)	3 (4.471)	1 (0.748)	-	3 (4.483)
	20'	6 (4.741)	2 (2.445)	0 (0.184)	4 (3.308)	7 (4.940)	3 (2.782)	1 (0.279)	-	5 (3.895)
*euclidEF*300	2'	5 (8.074)	3 (7.052)	0 (1.896)	4 (7.747)	7 (8.473)	4 (7.832)	1 (5.698)	-	2 (6.129)
	10'	6 (5.984)	2 (3.983)	0 (0.293)	2 (4.239)	7 (6.241)	4 (4.524)	1 (0.657)	-	4 (4.482)
	20'	6 (4.893)	2 (2.436)	0 (0.099)	4 (3.032)	7 (5.070)	3 (2.771)	1 (0.173)	-	5 (3.894)
*euclidAB*500	3'	4 (8.551)	3 (8.067)	0 (4.683)	7 (9.169)	6 (8.840)	5 (8.664)	2 (7.763)	-	1 (6.109)
	15'	6 (7.132)	3 (6.122)	0 (0.696)	3 (6.248)	7 (7.469)	5 (6.840)	1 (3.936)	-	2 (4.857)
	30'	6 (6.363)	3 (4.975)	0 (0.138)	2 (4.754)	7 (6.669)	5 (5.604)	1 (1.246)	-	2 (4.349)
*euclidCD*500	3'	4 (8.630)	3 (8.172)	0 (4.673)	7 (9.191)	6 (8.924)	5 (8.754)	2 (7.873)	-	1 (6.209)
	15'	6 (7.249)	3 (6.301)	0 (0.665)	3 (6.113)	7 (7.539)	5 (6.919)	1 (3.988)	-	2 (4.890)
	30'	6 (6.480)	4 (5.139)	0 (0.142)	2 (4.569)	7 (6.741)	5 (5.681)	1 (1.291)	-	2 (4.338)
*euclidEF*500	3'	4 (8.764)	2 (8.251)	0 (4.793)	7 (9.321)	6 (9.057)	5 (8.900)	2 (8.010)	-	1 (6.332)
	15'	6 (7.312)	4 (6.317)	0 (0.679)	3 (5.991)	7 (7.676)	5 (7.084)	1 (4.065)	-	2 (5.050)
	30'	6 (6.540)	4 (5.129)	0 (0.162)	2 (4.456)	7 (6.857)	5 (5.784)	1 (1.285)	-	2 (4.540)
TSP-3										
*euclidABC*100	1'	5 (5.314)	2 (4.263)	0 (2.629)	5 (5.481)	4 (5.101)	1 (3.996)	3 (4.726)	8 (9.832)	7 (6.527)
	5'	4 (2.330)	1 (1.212)	0 (1.066)	3 (1.816)	4 (2.209)	1 (1.174)	6 (2.569)	8 (9.463)	7 (5.653)
	10'	4 (1.432)	0 (0.741)	0 (0.653)	3 (1.230)	4 (1.342)	0 (0.685)	6 (1.772)	8 (8.915)	7 (5.394)
*euclidDEF*100	1'	5 (5.424)	2 (4.228)	0 (2.721)	5 (5.673)	4 (5.245)	1 (4.012)	3 (4.828)	8 (9.891)	7 (6.666)
	5'	4 (2.321)	0 (1.265)	0 (1.251)	3 (1.953)	4 (2.289)	0 (1.271)	6 (2.617)	8 (9.573)	7 (5.762)
	10'	3 (1.433)	0 (0.807)	0 (0.854)	3 (1.359)	4 (1.481)	0 (0.796)	6 (1.761)	8 (9.076)	7 (5.500)
*euclidABC*300	2'	4 (9.561)	3 (9.390)	0 (5.071)	7 (9.809)	6 (9.684)	5 (9.608)	1 (8.692)	-	1 (8.694)
	10'	5 (8.156)	2 (7.358)	0 (0.959)	4 (7.982)	7 (8.300)	3 (7.747)	1 (4.017)	-	5 (8.127)
	20'	5 (7.090)	2 (5.780)	0 (0.254)	2 (5.993)	6 (7.199)	3 (6.119)	1 (2.143)	-	7 (7.940)
*euclidDEF*300	2'	4 (9.506)	3 (9.365)	0 (5.108)	7 (9.802)	6 (9.661)	5 (9.590)	2 (8.656)	-	1 (8.562)
	10'	4 (8.021)	2 (7.325)	0 (1.059)	4 (8.102)	6 (8.251)	3 (7.686)	1 (4.016)	-	4 (8.000)
	20'	5 (6.969)	2 (5.748)	0 (0.321)	2 (5.863)	6 (7.093)	3 (6.035)	1 (2.231)	-	7 (7.841)
*euclidABC*500	3'	4 (9.794)	3 (9.739)	0 (7.429)	7 (9.928)	6 (9.857)	5 (9.836)	2 (9.599)	-	1 (8.687)
	15'	4 (9.158)	3 (8.920)	0 (1.303)	7 (9.488)	6 (9.339)	5 (9.184)	1 (6.453)	-	2 (8.210)
	30'	4 (8.497)	2 (8.008)	0 (0.248)	6 (8.683)	6 (8.703)	4 (8.386)	1 (3.636)	-	3 (8.051)
*euclidDEF*500	3'	4 (9.830)	3 (9.805)	0 (7.734)	7 (9.938)	6 (9.881)	5 (9.871)	2 (9.667)	-	1 (8.820)
	15'	4 (9.289)	3 (9.091)	0 (1.517)	7 (9.540)	6 (9.445)	5 (9.314)	1 (6.837)	-	2 (8.339)
	30'	5 (8.712)	3 (8.306)	0 (0.228)	5 (8.768)	7 (8.920)	4 (8.588)	1 (3.887)	-	2 (8.189)

Table 5 Average number of random restart operations for TSP benchmark test instances. The size of the best nondominated set for the instance under consideration is given as information in the second column ($|Z_N^\star|$).

Instance	$\lvert Z_N^\star \rvert$	DMLS $(1 \cdot 1)$	DMLS $(1 \cdot 1_{\not\prec})$	DMLS $(1 \cdot 1_{\succ})$	DMLS $(1 \cdot \star)$	DMLS $(\star \cdot 1)$	DMLS $(\star \cdot 1_{\not\prec})$	DMLS $(\star \cdot 1_{\succ})$	DMLS $(\star \cdot \star)$
TSP-2									
*euclidAB*100	1719	1.00	1.00	4.95	0.05	0.00	0.00	1.35	0.00
*euclidCD*100	2123	1.00	1.00	4.95	0.40	0.00	0.00	1.15	0.00
*euclidEF*100	2387	1.00	1.00	5.10	0.15	0.00	0.00	1.45	0.00
*euclidAB*300	8691	0.00	0.00	0.00	0.00	0.00	0.00	0.00	-
*euclidCD*300	8628	0.00	0.00	0.00	0.00	0.00	0.00	0.00	-
*euclidEF*300	8205	0.00	0.00	0.00	0.00	0.00	0.00	0.00	-
*euclidAB*500	16391	0.00	0.00	0.00	0.00	0.00	0.00	0.00	-
*euclidCD*500	14925	0.00	0.00	0.00	0.00	0.00	0.00	0.00	-
*euclidEF*500	14413	0.00	0.00	0.00	0.00	0.00	0.00	0.00	-
TSP-3									
*euclidABC*100	1155	1.00	1.00	1.80	0.05	0.00	0.00	0.00	0.00
*euclidDEF*100	1431	1.00	1.00	1.60	0.15	0.00	0.00	0.00	0.00
*euclidABC*300	588	0.00	0.00	0.00	0.00	0.00	0.00	0.00	-
*euclidDEF*300	480	0.00	0.00	0.00	0.00	0.00	0.00	0.00	-
*euclidABC*500	473	0.00	0.00	0.00	0.00	0.00	0.00	0.00	-
*euclidDEF*500	691	0.00	0.00	0.00	0.00	0.00	0.00	0.00	-

4.4 Further Comments

The experimental part has been conducted on two different multiobjective combinatorial optimization problems, the FSP and the TSP, both an a two-objective and a three-objective way. First of all, let us remark that the TSP and the FSP share some similarities since the feasible solutions for both problems can be represented as permutations. However, there also exists some differences between them. First, the objective functions for the TSP are from the same nature, whereas they are intrinsically correlated and of different kinds for the FSP (T'Kindt and Billaut, 2002). Second, as reported in Table 3 and Table 5, the size of the nondominated set is quite different from a problem to another. Indeed, identifying the whole efficient set for the TSP becomes computationally prohibitive for large-size problem instances, so that we generally have to deal with a bounded-size approximation. At last, even if both problems share a permutation-based encoding, the efficiency of problem-related search operators is not only related to the representation, but also to the type of problem to be solved (Talbi, 2009). For the TSP, the main focus relies on the adjacency of the elements contained in the permutation, whereas this is the relative order in the sequence that is important for the FSP. This is the reason why the neighborhood operator is not the same for the two problems. For instance, the 2-opt exchange operator is well-known to be efficient for the TSP because there exists a strong locality between the nodes, so that it generates small variations only. On the contrary, it is not very efficient for the FSP because it generates large variations.

From a purely design perspective, DMLS $(\star \cdot \star)$ seems to be the overall more relevant method if an unlimited amount of CPU time and memory space is assumed. Indeed, its search space exploration capability is likely the larger one when compared to other DMLS approaches. However, it becomes quickly limited in practice with respect to the size of the search space and of the Pareto front. DMLS $(1 \cdot \star)$ overcomes these scalability limitations by reducing time and space complexity. Indeed, it evaluates a limited subset of neighbors *per* iteration, and the number of exhaustive neighborhood examination is reduced. In practice, DMLS $(1 \cdot \star)$ globally obtains better results than DMLS $(\star \cdot \star)$, at least for large-size instances, and does not cause any scalability problem. On the other hand, and in opposition to the methods mentioned above, DMLS $(1 \cdot 1)$ and DMLS $(\star \cdot 1)$ first appear to be a bit too random. They provide good diversification abilities but present a lack of exploitation, as the intensification pressure is applied during the archiving step only. Our FSP results indicate their decent performance on problem instances where a large amount of efficient solutions are likely to be found. But, for the TSP, where the number of nondominated solutions is even higher, these approaches seem to fail due to the bounded archiving mechanism. Indeed, let us remind that solutions that are nondominated and nondominating at the same time are not allowed anymore when the archive is at its full capacity.

In consequence, while keeping the trade-off between intensification and diversification in mind, guiding the neighborhood exploration scheme by a dominance-based strategy appears to be more reliable. Firstly, in comparison to examining a random neighbor per current solution, the DMLS $(1 \cdot 1_{\not\prec})$ and DMLS $(\star \cdot 1_{\not\prec})$ algorithms offer equivalent diversification aptitudes. But they do not consume computational resources to handle dominated, and then unattractive solutions. Besides, restricting the acceptance of a neighboring solution increases the chance of getting an improving solution with regards to the current nondominated set, so that there is an indirect enhancement of exploration. The general trends of the results are quite similar to the ones reported for DMLS $(1 \cdot 1)$ and DMLS $(\star \cdot 1)$, but with a highest performance. At last, the 'first improving' strategy seems to be the ideal trade-off between the 'first nondominated' one and the exhaustive one, then providing an interesting exploration-exploitation compromise. Indeed, intensification abilities are clearly improved with respect

to both DMLS $(1 \cdot 1_{\not\prec})$ and DMLS $(\star \cdot 1_{\not\prec})$, while keeping diversification reasonable, as nondominated neighbors found during the neighborhood exploration process are still proposed for archiving. Furthermore, when compared to DMLS $(1 \cdot \star)$ and DMLS $(\star \cdot \star)$, the convergence to a locally efficient optimum set is accelerated. Although, the latter ones are more likely to move closer to the Pareto optimal front than former ones at a given iteration. In practice, DMLS $(1 \cdot 1_{\succ})$ and DMLS $(\star \cdot 1_{\succ})$ are the overall best-performing methods for almost the whole set of our experiments.

Now, with regards to current set selection, the experimental analysis does not allow to provide a clear conclusion on the strategy to adopt. However, it seems that selecting a single random solution is slightly more efficient, because corresponding DMLS approaches obtain statistically better results than their respective counterparts more often than the opposite.

5 Conclusion and Future Research

This paper investigated a class of local search methods for multiobjective combinatorial optimization. When applying local search methods to approximate a MCOP efficient set, two general approaches can generally be distinguished: scalar and dominance-based methodologies. We here focused on methods based on the successive improvement of the current set of nondominated solutions by means of a dominance relation. The first goal was to provide useful guidelines to practitioners on the design of such algorithms. Our attempt was to extend single-objective local search to multiobjective combinatorial optimization while matching its classical features as far as possible. Following a brief review of the literature, we identified the main search components for a better understanding of their working mechanisms. We discussed both problem-related and problem-independent components, and we identified different schemes, from very simple to more advanced ones. The general-purpose issues of dominance relation, current set selection, neighborhood exploration, archiving and stopping condition have been presented in more depth. We highlighted how existing approaches follow the same general methodology, and we proposed a general model unifying the design of DMLS methods, as well as a software package for their flexible implementation under the ParadisEO-MOEO framework. The model proposed in the paper provides a common description of DMLS algorithms, and allows the design of new techniques. At last, we studied the impact of some search components on solving different MCOPs: a two- and a three-objective FSP, as well as a two- and a three-objective TSP. In particular, we analyzed how relevant are existing and original current set selection and neighborhood exploration strategies on the overall algorithm efficiency. The experimental results were analyzed in a concise and sound way, following a rigorous statistical methodology for assessing the performance of multiobjective search methods.

The outcome of our experimental analysis gives valuable insight on the performance of different DMLS search components, at least for the MCOPs investigated in this paper. Firstly, it seems more accurate to guide the neighborhood exploration of a given solution by a clear dominance-based rule, and then to obtain a good trade-off between the evaluation of a single random neighbor and the evaluation of all neighbors. Indeed, our results are unambiguous and indicates that the neighborhood exploration of a current solution x should stop either once a nondominated neighbor, or once a dominating one, with respect to x, is found. With such strategies, the number of neighbor evaluations is somehow intrinsically chosen adaptively, depending on the current solution quality and then its presumed ability to find potentially interesting neighbors. Indeed, it should be more difficult to find a dominating neighbor at the end of the search, given that a current solution is supposed to be closer to the

Pareto front. Nevertheless, the choice of continuing the exploration until a nondominated neighbor, or a dominating one, is found is not that clear. However, it seems to be closely related to the size of the problem to be solved, to the number of objective functions to handle, and to the number of efficient solutions that are likely to be found. Though, from our set of experiments, it seems that the 'first dominating neighbor' strategy is the overall more efficient one. Secondly, the choice of the number of solutions to be explored at a given iteration does not seem to have a major influence on the algorithm performance. Indeed, selecting the whole current nondominated set or a single item at random globally obtain similar results, even if a partial selection seems to perform slightly better. Furthermore, note that, depending on the implementation, selecting the set of archived solutions exhaustively could appear impracticable for large-size problems. It could still be interesting to decide a good trade-off between a single and the whole set of solutions to select.

A number of issues are open for future investigation. Some of them are discussed below.

- Further research is required to experimentally analyze the influence of other DMLS search components on the overall algorithm behavior. More advanced problem-related components would surely improve the results, especially in terms of initialization and neighborhood operator, as already shown by Paquete and Stützle (2009) for the TSP. For instance, we already know that the performance of PLS-1 can be largely improved while starting with a number of good-quality solutions, as empirically shown by recent studies (Paquete and Stützle, 2006; Paquete and Stützle, 2009; Dubois-Lacoste et al, 2009; Lust and Teghem, 2010; Lust and Jaszkiewicz, 2010).
- Problem-independent issues, including archiving, might also be subject to investigation in the frame of DMLS algorithmic design, even if archiving techniques have already been subject to discussion in the specialized literature (Knowles and Corne, 2004). Sophisticated archiving techniques are generally time-consuming, whereas it is always a hard task to find the ideal trade-off between computational time and solution quality. A good alternative would be to base the archiving rule on ε-dominance, as proposed by Laumanns et al (2002). But the setting of a proper ε-value generally requires a deep knowledge about the problem to be solved in order to obtain a suitable archive size.
- Another issue is the analysis of the influence of DMLS components depending on the search space characteristics, including connectedness between efficient solutions (Ehrgott and Klamroth, 1997).
- Further experiments of DMLS algorithms should also be lead for solving other MCOPs, including real-world applications. In particular, we believe that DMLS $(1 \cdot 1_{\succ})$ could offer an attractive search method. Indeed, DMLS algorithms already offer a high level of simplicity while requiring a very small number of parameters. This is the reason why the performance of DMLS methods should also be compared to state-of-art multiobjective evolutionary algorithms, that are known to handle a highest number of parameters. As well, designing hybrid metaheuristics between both would offer an alternative to existing ones that often handle scalar local search methods (Ehrgott and Gandibleux, 2008).
- Advanced DMLS variants, based on tabu search or simulated annealing, might also be explored, together with parallel and distributed versions. Finally, extending and generalizing the DMLS model proposed in the paper to other existing multiobjective local search methods would provide a powerful methodology for the design of efficient multiobjective search methods.

Acknowledgements The authors would like to acknowledge the anonymous reviewers for their valuable comments and suggestions that largely contributed to improve the quality of the paper.

References

Aguirre H, Tanaka K (2005) Random bit climbers on multiobjective MNK-landscapes: Effects of memory and population climbing. IEICE Trans Fundamentals 88-A(1):334–345

Angel E, Bampis E, Gourvés L (2004) A dynasearch neighbohood for the bicriteria traveling salesman problem. In: Metaheuristics for Multiobjective Optimisation, Lecture Notes in Economics and Mathematical Systems, vol 535, Springer-Verlag, Berlin, Germany, chap 6, pp 153–176

Basseur M, Seynhaeve F, Talbi EG (2002) Design of multi-objective evolutionary algorithms: Application to the flow shop scheduling problem. In: IEEE Congress on Evolutionary Computation (CEC 2002), IEEE Press, Piscataway, NJ, USA, pp 1151–1156

Bleuler S, Laumanns M, Thiele L, Zitzler E (2003) PISA — a platform and programming language independent interface for search algorithms. In: Second International Conference on Evolutionary Multi-Criterion Optimization (EMO 2003), Springer-Verlag, Faro, Portugal, Lecture Notes in Computer Science, vol 2632, pp 494–508

Brockhoff D, Friedrich T, Hebbinghaus N, Klein C, Neumann F, Zitzler E (2007) Do additional objectives make a problem harder? In: Genetic and Evolutionary Computation Conference (GECCO 2007), ACM Press, New York, NY, USA, pp 765–772

Coello Coello CA, Lamont GB, Van Veldhuizen DA (2007) Evolutionary Algorithms for Solving Multi-Objective Problems (second edition). Genetic and Evolutionary Computation Series, Springer, New York, USA

Davis L (1985) Job shop scheduling with genetic algorithms. In: First International Conference on Genetic Algorithms (ICGA 1985), L. Erlbaum Associates Inc., Hillsdale, NJ, USA, pp 136–140

Deb K (2001) Multi-Objective Optimization using Evolutionary Algorithms. John Wiley & Sons, Chichester, UK

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

Dubois-Lacoste J, López-Ibáñez M, Stützle T (2009) Effective hybrid stochastic local search algorithms for biobjective permutation flowshop scheduling. In: International Workshop on Hybrid Metaheuristics (HM 2009), Lecture Notes in Computer Science, vol 5818, Springer-Verlag, Berlin, Germany, pp 100–114

Ehrgott M, Gandibleux X (2004) Approximative solution methods for multiobjective combinatorial optimization. TOP 12(1):1–89

Ehrgott M, Gandibleux X (2008) Hybrid metaheuristics for multi-objective combinatorial optimization. In: Hybrid Metaheuristics: An Emerging Approach to Optimization, Studies in Computational Intelligence, vol 114, Springer-Verlag, Berlin, Germany, chap 8, pp 221–259

Ehrgott M, Klamroth K (1997) Connectedness of efficient solutions in multiple criteria combinatorial optimization. European Journal of Operational Research 97(1):159–166

Gandibleux X, Mezdaoui N, Fréville A (1997) A tabu search procedure to solve multiobjective combinatorial optimization problems. In: Advances in Multiple Objective and Goal Programming, Lecture Notes in Economics and Mathematical Systems, vol 455, Springer-Verlag, Berlin, Germany, pp 291–300

Geiger M (2007) On operators and search space topology in multi-objective flow shop scheduling. European Journal of Operational Research 181(1):195–206

Graham RL, Lawler EL, Lenstra JK, Rinnooy Kan AHG (1979) Optimization and approximation in deterministic sequencing and scheduling: A survey. Annals of Discrete Mathematics 5:287–326

Hansen MP (1997) Tabu search for multiobjective optimisation : MOTS. In: Thirteenth International Conference on Multiple Criteria Decision Making (MCDM 1997), Springer-Verlag, Cape Town, South Africa

Ishibuchi H, Murata T (1998) A multi-objective genetic local search algorithm and its application to flowshop scheduling. IEEE Transactions on Systems, Man and Cybernetics 28:392–403

Jaszkiewicz A (2002) Genetic local search for multi-objective combinatorial optimization. European Journal of Operational Research 137(1):50–71

Knowles J, Corne D (2004) Bounded Pareto archiving: Theory and practice. In: Metaheuristics for Multiobjective Optimisation, Lecture Notes in Economics and Mathematical Systems, vol 535, Springer-Verlag, Berlin, Germany, chap 2, pp 39–64

Knowles J, Thiele L, Zitzler E (2006) A tutorial on the performance assessment of stochastic multiobjective optimizers. TIK Report 214, Computer Engineering and Networks Laboratory (TIK), ETH Zurich, Zurich, Switzerland, (revised version)

Knowles JD, Corne D (2000) Approximating the nondominated front using the Pareto archived evolution strategy. Evolutionary Computation 8(2):149–172

Landa Silva JD, Burke E, Petrovic S (2004) An introduction to multiobjective metaheuristics for scheduling and timetabling. In: Metaheuristics for Multiobjective Optimisation, Lecture Notes in Economics and Mathematical Systems, vol 535, Springer-Verlag, Berlin, Germany, pp 91–129

Laumanns M, Thiele L, Deb K, Zitzler E (2002) Combining convergence and diversity in evolutionary multi-objective optimization. Evolutionary Computation 10(3):263–282

Laumanns M, Thiele L, Zitzler E (2004) Running time analysis of evolutionary algorithms on a simplified multiobjective knapsack problem. Natural Computing 3(1):37–51

Liefooghe A, Basseur M, Jourdan L, Talbi EG (2007) Combinatorial optimization of stochastic multi-objective problems: an application to the flow-shop scheduling problem. In: Fourth International Conference on Evolutionary Multi-criterion Optimization (EMO 2007), Springer-Verlag, Matsushima, Japan, Lecture Notes in Computer Science, vol 4403, pp 457–471

Liefooghe A, Jourdan L, Legrand T, Humeau J, Talbi EG (2010) ParadisEO-MOEO: A software framework for evolutionary multi-objective optimization. In: Advances in Multi-Objective Nature Inspired Computing, Studies in Computational Intelligence, vol 272, Springer, Berlin, Germany, chap 5, pp 87–117

Liefooghe A, Jourdan L, Talbi EG (2011) A software framework based on a conceptual unified model for evolutionary multiobjective optimization: ParadisEO-MOEO. European Journal of Operational Research 209(2):104–112

Lust T, Jaszkiewicz A (2010) Speed-up techniques for solving large-scale biobjective TSP. Computers & Operations Research 37(3):521–533

Lust T, Teghem J (2010) Two-phase pareto local search for the biobjective traveling salesman problem. Journal of Heuristics 16(3):475–510

Miettinen K (1999) Nonlinear Multiobjective Optimization, International Series in Operations Research and Management Science, vol 12. Kluwer Academic Publishers, Boston, MA, USA

Neumann F, Wegener I (2006) Minimum spanning trees made easier via multi-objective optimization. Natural Computing 5(3):305–319

Paquete L, Stützle T (2003) A two-phase local search for the biobjective traveling salesman problem. In: Second International Conference on Evolutionary Multi-Criterion Optimization (EMO 2003), Springer-Verlag, Faro, Portugal, Lecture Notes in Computer Science, vol 2632, pp 479–493

Paquete L, Stützle T (2006) A study of stochastic local search algorithms for the biobjective QAP with correlated flow matrices. European Journal of Operational Research 169(3):943–959

Paquete L, Stützle T (2007) Stochastic local search algorithms for multiobjective combinatorial optimization: A review. In: Gonzalez TF (ed) Handbook of Approximation Algorithms and Metaheuristics, Computer & Information Science Series, vol 13, Chapman & Hall / CRC

Paquete L, Stützle T (2009) Design and analysis of stochastic local search for the multiobjective traveling salesman problem. Computers & Operations Research 36(9):2619–2631

Paquete L, Chiarandini M, Stützle T (2004) Pareto local optimum sets in the biobjective traveling salesman problem: An experimental study. In: Metaheuristics for Multiobjective Optimisation, Lecture Notes in Economics and Mathematical Systems, vol 535, Springer-Verlag, Berlin, Germany, chap 7, pp 177–199

Paquete L, Schiavinotto T, Stützle T (2007) On local optima in multiobjective combinatorial optimization problems. Annals of Operations Research 156(1):83–97

Rudolph G, Agapie A (2000) Convergence properties of some multi-objective evolutionary algorithms. In: IEEE Congress on Evolutionary Computation (CEC 2000), IEEE Press, Piscataway, New Jersey, USA, pp 1010–1016

Serafini P (1986) Some considerations about computational complexity for multiobjective combinatorial problems. In: Recent advances and historical development of vector optimization, Springer-Verlag, Berlin, Germany, Lecture Notes in Economics and Mathematical Systems, vol 294, pp 222–231

Serafini P (1992) Simulated annealing for multiobjective optimization problems. In: Tenthth International Conference on Multiple Criteria Decision Making (MCDM 1992), Taipei, Taiwan, pp 87–96

Taillard ED (1993) Benchmarks for basic scheduling problems. European Journal of Operational Research 64:278–285

Talbi EG (2009) Metaheuristics: from design to implementation. Wiley

Talbi EG, Rahoual M, Mabed MH, Dhaenens C (2001) A hybrid evolutionary approach for multicriteria optimization problems : Application to the fow shop. In: First International Conference on Evolutionary Multi-criterion Optimization (EMO 2001), Springer-Verlag, Zurich, Switzerland, Lecture Notes in Computer Science, vol 1993, pp 416–428

T'Kindt V, Billaut JC (2002) Multicriteria Scheduling: Theory, Models and Algorithms. Springer-Verlag, Berlin, Germany

Ulungu EL, Teghem J, Fortemps P, Tuyttens D (1999) MOSA method: a tool for solving multiobjective combinatorial optimization problems. Journal of Multi-Criteria Decision Analysis 8(4):221–236

Zitzler E, Laumanns M, Thiele L (2001) SPEA2: Improving the strength pareto evolutionary algorithm. TIK Report 103, Computer Engineering and Networks Laboratory (TIK), ETH Zurich, Zurich, Switzerland

Zitzler E, Thiele L, Laumanns M, Foneseca CM, Grunert da Fonseca V (2003) Performance assessment of multiobjective optimizers: An analysis and review. IEEE Transactions on Evolutionary Computation 7(2):117–132