Computers & Operations Research 39 (2012) 3089–3101

Contents lists available at SciVerse ScienceDirect

Computers & Operations Research

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

Multi-directional local search

Fabien Tricoire [a,b,*]

[a] Department of Business Administration, University of Vienna, Bruenner Strasse 72, 1210 Wien, Austria
[b] NICTA/UNSW, Sydney, Australia

ARTICLE INFO

Available online 27 March 2012

Keywords:
Multi-objective optimization
Metaheuristics

ABSTRACT

This paper introduces multi-directional local search, a metaheuristic for multi-objective optimization. We first motivate the method and present an algorithmic framework for it. We then apply it to several known multi-objective problems such as the multi-objective multi-dimensional knapsack problem, the bi-objective set packing problem and the bi-objective orienteering problem. Experimental results show that our method systematically provides solution sets of comparable quality with state-of-the-art methods applied to benchmark instances of these problems, within reasonable CPU effort. We conclude that the proposed algorithmic framework is a viable option when solving multi-objective optimization problems.

1. Introduction

Despite indisputable progress in combinatorial optimization in the last 60 years, there still exists a gap between theory and practice; for instance, when confronted with hard combinatorial problems, decision makers are not always able to formulate them directly as linear or mixed integer-linear programs, not to mention solving them. There are several reasons for this, one of them being that many real-world problems are in fact multi-objective. This is typically the case when a decision maker wants to optimize several aspects of his problem, while being unable to express preferences among these objectives.

Many real-world optimization problems have several objectives and, in the last decade, research studies focusing on multi-objective problems became more common. This has been made possible by both technological progress (faster computers) and the construction of necessary theoretical tools for multi-objective optimization: although it is simple to compare two solutions when considering a single objective, it becomes more complicated when considering several objectives. A common approach, called the Pareto approach, is to consider a dominance rule such that a solution dominates another one if it is better in at least one objective and not worse in all objectives. A solution is Pareto-optimal if there does not exist a solution that dominates it. In such a context, the goal of an optimization algorithm is to find not only one optimal solution, but the whole set of Pareto-optimal solutions. This set can be partitioned into supported and non-supported solutions. For any k-objective problem and any given weight vector of size k, there exists a single-objective projected problem obtained by performing a linear combination of all weighted objectives. Supported solutions are then defined as those solutions for which there exists a weight vector, with strictly positive weights, such that they are optimal for the associated single-objective projected problem.

Although only one solution can be implemented in practice, the Pareto approach has advantages when the decision maker's preference is not known a priori: no matter what this preference is, the solution that will be optimal under this preference is a Pareto-optimal solution, and providing the Pareto front can help eliciting the decision maker preference by presenting them a set of trade-off solutions. This is called the *a posteriori approach* [54]. In some cases the set of trade-off solutions can grow very large, so the decision maker must be assisted in exploring it. Interactive analysis methods, such as the interactive trichotomy [30], aim at providing such assistance. An overview of these interactive methods is provided by Jaszkiewicz and Branke [29]. The authors note a trend in integrating the two stages, thus allowing the decision maker to interfere in the search process in order to prune the solution space during the optimization. For instance, Branke et al. [5] integrate the GRIP decision analysis method [18] with NSGA-II [12].

Although decision analysis methods continuously improve and provide new mechanisms, they still rely on the ability to identify Pareto fronts efficiently. For that reason, Pareto optimization is a relevant research problem in relation to multi-criteria decision making.

For reasons including computational power limitations, heuristics and metaheuristics are a frequent choice when it comes to optimizing multiple objectives. In particular, evolutionary algorithms are quite

* Corresponding author at: Department of Business Administration, University of Vienna, Bruenner Strasse 72, 1210 Wien, Austria.
E-mail addresses: fabien.tricoire@univie.ac.at, fabien@nicta.com.au

0305-0548
http://dx.doi.org/10.1016/j.cor.2012.03.010

popular. This dates back to the seminal contribution by Schaffer [45]. Since then, numerous evolutionary algorithms for multi-objective optimization have been developed, like for instance the non-dominated sorting genetic algorithm (NSGA [49]), the genetic algorithms by Coello and Christiansen [8], the multi-objective genetic algorithm (MOGA [21]), the Pareto archived evolution strategy (PAES [32]), the non-dominated sorting genetic algorithm (NSGA-II [12]), or the strength Pareto evolutionary algorithm 2 (SPEA2 [57]). In general, an important contribution of such methods lies in the adaptation of the concept of *fitness* to the multi-objective context. For instance, in NSGA and NSGA-II the population is sorted into different fronts (or levels): front 1 contains the non-dominated solutions, front 2 the solutions that are only dominated by solutions of front 1, and so on. The fitness of a solution is then the front level to which it belongs. With SPEA2, the fitness of a solution is based on its *strength*, which represents the number of individuals it dominates. With several popular multi-objective evolutionary algorithms, like for instance NSGA-II or SPEA2, dominated solutions are kept under certain conditions, usually in order to bring diversity. The reader interested in a deeper understanding of multi-objective evolutionary algorithms will find a good starting point in the book by Coello Coello et al. [9].

Other traditional metaheuristics have been adapted to multi-objective optimization as well. Simulated annealing was the basis for various contributions (e.g. [53,10,26]), as well as tabu search (e.g. [24,23]), a mix of both [2], ant colony optimization and variable neighborhood search [15,41,46] to name a few. As already discussed with evolutionary algorithms, adapting the concept of solution quality to the multi-objective context is a major issue when developing multi-objective heuristic methods. One possibility is to use weighted-sum functions, but it has the drawback that non-supported solutions are not captured by such a projection. As an alternative, weighted Tchebycheff scalarizing functions can be used (see e.g. [1]). Some algorithms still rely on weighted-sum functions. For instance Parragh et al. [41] first solve various weighted-sum single-objective versions of the dial-a-ride problem (using variable neighborhood search), then search for non-supported solutions using path relinking.

Path relinking is a common ingredient of multi-objective optimization methods. It consists in, from an initial solution, performing moves to reach a guiding solution, thus linking both solutions with a *path*, in which each step is associated with an intermediate solution. The underlying expectation is that these intermediate solutions might be Pareto-optimal, provided both initial and guiding solutions are good enough. For an introduction to path relinking, the interested reader should refer to Glover and Laguna [25].

A generalization of local search to the multi-objective context, called Pareto local search (PLS), has been proposed by Paquete et al. [39]. It was further analyzed by Paquete et al. [40]. The concept of local optimum is extended by Paquete et al. [39] to the multi-objective context by two definitions: (i) a solution p is a Pareto local optimum with respect to neighborhood N if there exists no $r \in N(p)$ such that r dominates p, and (ii) P is a Pareto local optimum set with respect to neighborhood N if and only if it contains only Pareto local optima with respect to N. Building up on these definitions, PLS is defined as a procedure that iteratively improves a non-dominated set through neighborhood exploration. More precisely, solutions are stored in an archive, and at each iteration a solution's neighborhood is explored, which leads to updating the archive. Each solution's neighborhood is explored only once, and the algorithm stops when the neighborhoods of all solutions in the archive have been processed; the archive is then a Pareto local optimum set.

This paper presents a new metaheuristic for multi-objective optimization, called multi-directional local search (MDLS). MDLS only keeps track of non-dominated solutions, like PLS, but unlike many well-known methods for multi-objective optimization. It can be seen as a multi-objective generalization of stochastic local search (SLS [27]).

In Section 2 we introduce the method and describe its key ideas and general framework. In Section 3 we describe our experimental protocol, including the MDLS implementation. In Section 4 we validate it through experimentation by applying it to three multi-objective optimization problems, comparing the results with benchmark data from the literature. We conclude the paper with propositions for future research on multi-directional local search, and identify promising venues to broaden the validation of the method to a range of problems.

2. Multi-directional local search

In this section, we present the new algorithmic framework MDLS. We first provide some overview on multi-objective metaheuristics as well as motivation for this method. We then present the algorithmic framework for MDLS.

2.1. Rationale

When designing a metaheuristic, one should keep in mind to provide a general-purpose method, regardless of problem-specific ingredients: while heuristics are designed to solve a given problem, the purpose of metaheuristics is to solve a more general *meta-problem*. Problem-specific heuristics are usually specialized components of such metaheuristics. In the context of multi-objective optimization the usual preoccupations still matter (e.g. diversification and intensification), but the meta-problem we want to tackle is, given a multi-objective optimization problem, to provide a good non-dominated front (and not only a good solution). Existing multi-objective metaheuristics typically rely on theoretical and/or empirical knowledge about characteristics of exact Pareto fronts, and aim at exploiting various properties in order to provide good approximation of the Pareto front.

A first example is the use of weighted sums, present in many local search based multi-objective metaheuristics: it relies on the fact that the optimum of a single-objective problem using a weighted sum (with strictly positive weights) for the objective is also part of the Pareto set. It is therefore likely that a good metaheuristic for the single-objective problem will produce good solutions for the multi-objective problem. An interesting property of that approach is that given two solutions x_1 and x_2, it holds that if x_1 is better than x_2 for at least one weighted sum function, then x_2 does not dominate x_1. A direct consequence is that all traditional single-objective methods can be directly used with such weighted sums. However, only the supported solutions of the Pareto set are also optimal for a weighted sum objective.

A second example is that of evolutionary multi-objective metaheuristics. Such methods rely heavily on the fact that combining together elements of *good* solutions may help to produce other good solutions. In the single-objective case the term *good* refers directly to the fitness function, and in the multi-objective case it refers to the ability of a given solution to dominate others. Another key idea of such algorithms is that in order to make it possible to provide efficient exploration of the solution space, diversity has to be maintained in the population.

The very definition of multi-objective optimization contains a simple theoretical tool that can be used in the design of heuristic algorithms: the concept of Pareto dominance. If, from a solution x, efficient or not, we want to find a neighbor x' which is efficient, then this means that we are looking for solutions that are either (i) dominating x or (ii) non-comparable with x. In both cases, this means that x' has to be better than x for at least one objective. So in order to find new efficient solutions that are neighbors of x, it is sufficient to search only in one *direction* at a time, i.e. to use single-objective local search. This principle is illustrated in Fig. 1,

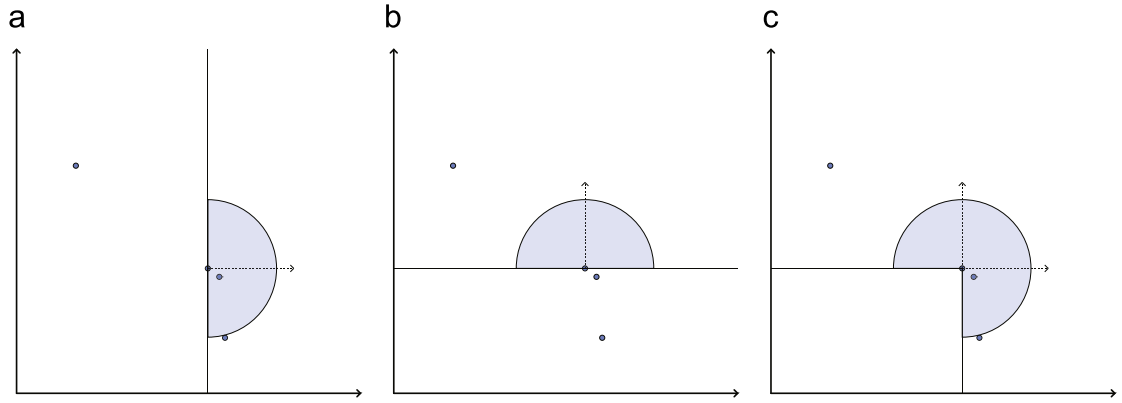

Fig. 1. Relevant portions of solution space for a bi-objective maximization problem. (a) Relevant portion of solution space in direction 1. (b) Relevant portion of solution space in direction 2. (c) Overall relevant portion of solution space.

with an example on a bi-objective maximization problem. Search direction 1 is illustrated in Fig. 1(a): the objective space around a given solution is cut in two halves, and the relevant half is the right one. The circular portion represents a neighborhood around the given solution. Fig. 1(b) represents the relevant objective space for direction 2 in a similar fashion. Fig. 1(c) represents the total relevant objective space around this same solution.

With these ideas in mind, we develop a method which consists in, given a non-dominated set of solutions, iteratively improving it by exploring neighborhoods using single-objective local search.

2.2. Algorithmic framework for multi-objective optimization

A key idea of MDLS is to use different local searches, each of them working on a single objective. More precisely, for each objective k a local search LS_k is defined. This local search is later performed in order to improve solutions with respect to objective k. An iteration consists in (i) selecting a solution, (ii) performing local search on this solution for each objective/direction, thus producing a new solution in each direction and (iii) accepting or rejecting the newly produced solutions. A requirement for the local search procedures used in the second step is that they only produce feasible solutions.

A risk when solving multi-objective problems with a local search based on a single incumbent solution is to miss whole regions of the Pareto front. To address this issue, we consider an incumbent set of solutions instead (also called *archive*). More precisely, a set F of non-dominated solutions is maintained, which contains all solutions found during the search that are non-dominated. This set is also what the MDLS returns when it stops.

Algorithm 1 gives an outline of MDLS, assuming a problem with K objectives.

Algorithm 1. Multi-Directional Local Search for K objectives.

```
1:  input: a set of non-dominated solutions F
2:  repeat
3:    x ← select_a_solution(F)
4:    G ← ∅
5:    for k ← 1 to K do
6:      G ← G ∪ {LS_k(x)}
7:    end for
8:  update(F,G)
9:  until stopping criterion is met
10: return F
```

We note similarities with PLS [39]: at each iteration a solution is selected from the archive and the non-dominated set is updated through neighborhood exploration around this solution. However there are sensible differences: PLS requires complete neighborhood exploration around a solution, whereas MDLS performs any kind of local search, including stochastic local search methods; MDLS is based on the previously introduced concept of search direction; more generally, PLS is deterministic (and is proved to converge) whereas MDLS is stochastic.

In fact, various aspects of MDLS can be found in other methods. For instance, the idea of selecting solutions from an archive, applying local search and updating the archive is not new. An example of methods applying a similar idea can be found in multi-objective iterated greedy search (see e.g. [22,37]). There are also other methods using single-objective local search, like the simulated annealing method of Varadharajan and Rajendran [55]. However, we believe that the core idea of MDLS, i.e. selecting a solution, searching around it in each direction then updating the archive, is new.

Regarding neighborhood exploration, any local search method can be used; in particular, any stochastic local search [27] can be used. Such methods are problem-specific components, and should be designed depending on the single-objective problem at hand.

In order to provide a concrete insight on how MDLS is designed to work and how it works in practice, we now present a graphical example on a bi-objective maximization problem in Fig. 2. It shows the first iterations of our MDLS implementation on the multi-objective orienteering problem described in Section 4.3, on instance squ_t070. Fig. 2(a) displays the starting set of solutions, at the beginning of the search. Fig. 2(b) shows the two neighbors obtained during iteration 1, one for each direction (objective). Fig. 2(c) presents the set of solutions at the end of iteration 1, after dominance checks. Fig. 2(d) shows the two neighbors obtained during iteration 2, again one for each optimization direction. Fig. 2(e) shows the set of solutions after dominance checks at iteration 2. Finally, Fig. 2(f) shows the set of solutions a few iterations later, at the end of iteration 10.

3. Experimental setting

In this section we provide more details regarding our implementation of multi-directional local search, and how we evaluate the quality of the solution sets produced. The algorithm presented in the previous section is implemented as-is; however, some components still need to be specified. In the following subsection, we present the common components for MDLS; in the subsequent subsection, we describe the performance indicators used to assess the quality of the method.

3.1. Common components for MDLS

We first note here that the archive F can grow very large in some cases; it is therefore important to use an appropriate data

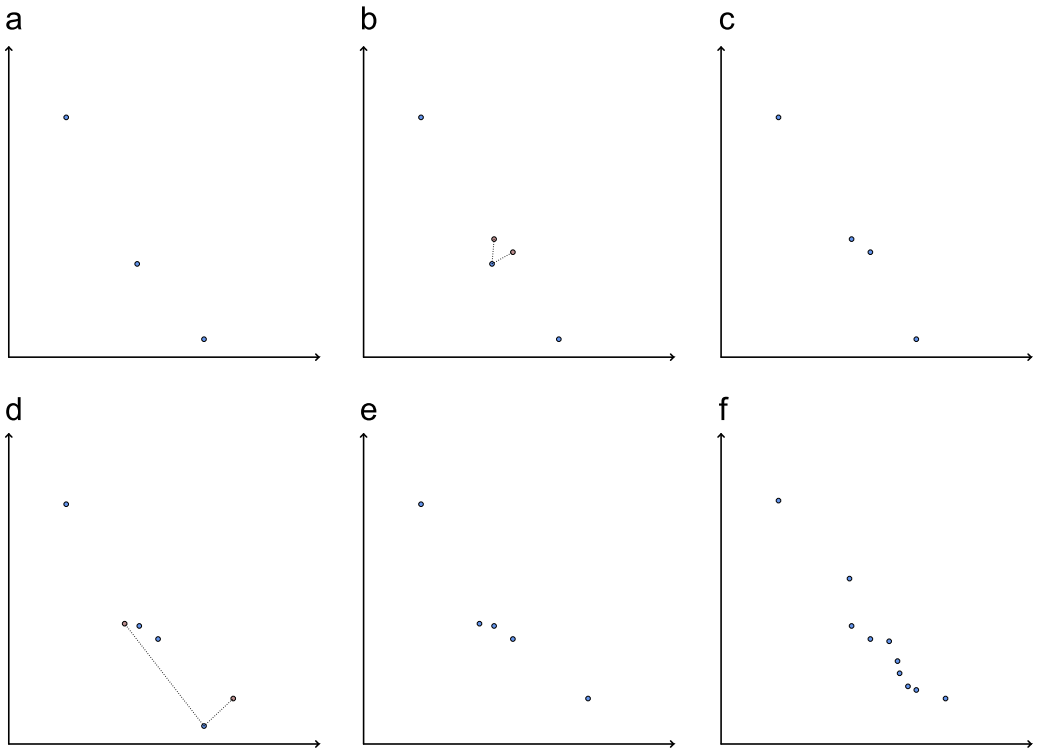

Fig. 2. First steps of MDLS on a bi-objective maximization problem. (a) Starting set of solutions. (b) Neighbors obtained during iteration 1. (c) Set of solutions at the end of iteration 1. (d) Neighbors obtained during iteration 2. (e) Set of solutions at the end of iteration 2. (f) Set of solutions at the end of iteration 10.

structure. In the context of MDLS, there are three operations that we want to perform on F:

1. Test whether solutions in F are dominated by a new solution s.
2. Test whether this new solution s is dominated by solutions in F.
3. Select a solution from F.

The first two operations are typically time-consuming and special care should be taken in order to perform them as efficiently as possible.

Perhaps the simplest way to represent a non-dominated set is by using an unordered list. When doing so, $|F|$ dominance tests have to be performed for operation 1, and $|F|$ more for operation 2. When considering a bi-objective maximization problem, it is possible to simply sort the list using decreasing values of first objective to compare solutions. As a side-effect, the list will be sorted in increasing values for the second objective. By doing so, we can consider two interesting subsets of F: the right part of the list corresponds to the subset of solutions that have a chance of being dominated by s, while the left part of the list contains the solutions that have a chance of dominating s. If there is a solution in F that has a similar value as s for the first objective then it is in both subsets. By sorting the list, the number of required dominance tests for every new solution is divided by two. This trick only works for two objectives.

A more general principle, quad-trees allow to partition F into three subsets: the solutions that have a chance of dominating s, the solutions that have a chance of being dominated by s, and the solutions that we know are non-comparable with s. For a detailed explanation on how quad-trees work, as well as variants of quad-trees, see Mostaghim and Teich [38]. Quad-trees offer a powerful partitioning of the non-dominated set, and work independently of the number of objectives considered.

In the context of this paper we use quad-trees to represent non-dominated sets. More precisely, we use the quad-tree 1 structure described by Mostaghim and Teich [38].

There are many ways to select a solution from F in order to guide the search, but in the following we always select it randomly, with equal probabilities for each element of F; this way, no parameter is required. In order to do so, we enrich the quad-tree structure by storing, at each node, the number of solutions in the corresponding sub-tree. This allows us to select random solutions from the quad-tree, by weighting the probability to select each node using the number of solutions it represents.

In order to stop the search any criterion can be used, but we always use the same one, which is to reach a predefined number of iterations. This number of iterations varies depending on the problem though, and is similar to CPU budget.

As explained in Section 2.2, any single-objective local search method can be used for directional search in MDLS. In the context of this paper, we use large neighborhood search (LNS) for that purpose. Different neighborhoods are used for the different problems we tackle, but they all follow a basic LNS scheme. We now summarize a few facts about LNS.

Large neighborhood search, also called *ruin and recreate*, is a metaheuristic introduced by Shaw [48]. The name "Large Neighborhood Search" stems from the fact that all solutions that can be reached by successively destroying a part of the incumbent solution then reconstructing it in a different way define an implicit neighborhood around it. Such a neighborhood is considered to be *large*. Since an exhaustive exploration of such neighborhoods would be too time-consuming, heuristics are used in an aggressive manner in order to narrow the search: the ruin and recreate operators typically aim at exploring only interesting parts of the implicit neighborhood. So far, LNS has been used almost exclusively to solve transportation problems. Shaw uses it in a constraint programming framework in order to solve the vehicle routing problem with time windows (VRPTW). Two years later, a similar principle is introduced by Schrimpf et al. [47] under the name *ruin and recreate*, and applied to the traveling salesman problem, the VRPTW, and network optimization. Pisinger and Ropke [42] develop a general-purpose

LNS method for a wide range of routing problems including capacitated vehicle routing problem, VRPTW, site-dependent vehicle routing problem, multi-depot vehicle routing problem, and open vehicle routing problem. A generalization of all these problems is considered, and each single problem is converted to the generalized version, then solved with the same LNS method. Experimental results are provided for standard benchmark instances on each of the five considered problems, which show LNS to outperform previous approaches. In an earlier research, Ropke and Pisinger [44] present an adaptive version of LNS in which the heuristic is dynamically tuned to select appropriate neighborhoods, and use it to solve the pickup and delivery problem with time windows. All these contributions and more are reviewed in the recent survey by Pisinger and Ropke [43].

In the context of MDLS, single-objective local search consists of one or several iterations of LNS. Therefore, for each objective of the problem at hand, a set of *ruin* and *recreate* operators must be designed. Such operators, which define neighborhoods, are problem-specific and are described later on in the respective subsection of each problem. The selection of which neighborhood to use at a given LNS iteration can be performed in several ways. For instance, Ropke and Pisinger [44] use an adaptive mechanism that emphasizes the use of the most successful neighborhoods. In this paper we use a very simple mechanism: the neighborhood is selected randomly, each neighborhood being equiprobable.

3.2. Performance indicators

In multi-objective optimization, one cannot simply compare the quality of two solutions. In order to provide a fair evaluation of the performance of various methods, indicators are used. Such indicators take as input the set of solutions provided by a method, and return a value reflecting one aspect of this set. Another part of the input can typically be a reference set to compare to. We now briefly present all performance indicators which are used to compare MDLS with other methods in the following sections. All these indicators are usually computed on normalized objective values, so that all objectives take values within the same range when considering the whole Pareto front. For detailed information on performance indicators for multi-objective optimization, we recommend the tutorial by Knowles et al. [33].

The hypervolume indicator, introduced by Zitzler and Thiele [58], represents the size of the space covered by a set of solutions. In the original paper it is actually called *size of the space covered*. For this indicator, a *nadir point* must be used in order to bound the space that is being measured. With this indicator, larger values are better.

The multiplicative unary epsilon indicator, introduced by Zitzler et al. [59], gives an indication on how far from a reference set R is an approximation set A. More precisely, and assuming all objectives should be minimized, it defines the smallest factor ϵ such that for each element $e \in R$ and its objective values vector v_e, $\epsilon \cdot v_e$ is dominated by at least one element in A. The smallest value for this indicator is 1, and smaller values are better.

The percentage of Pareto-optimal solutions found, called M1 [53], is also used. Highest values are the best, 1 being the maximum value (that of the Pareto set).

The convergence metric [31] gives the average Euclidean distance from an approximation front to a reference front, ideally to the Pareto front. For each point in the approximation set, the Euclidean distance to the closest solution in the reference set is calculated, then the average of these produces the indicator value. The best value is 0, and smaller values are better.

Although they all provide information of some aspect of the examined non-dominated set, these indicators all have drawbacks, and none of them should be used as a single measurement of quality. However by using several indicators at the same time one can provide a more objective comparison of two different methods. We consider that the joint use of the hypervolume and unary epsilon indicators provides a fair enough point of view when comparing two multi-objective methods; in the following, we systematically use these two indicators whenever possible. However, detailed experimental data from other research works is not always available. In such cases, we use the same indicators as those presented in the research works we compare to.

A concept related to performance indicators is that of *attainment sets* [11]. An attainment set is a set of solutions that are *attained* with a certain probability. A solution is attained by a set when there is another solution in this set that dominates it. For instance, the 50% attainment set of a given method on a given instance is the set of solutions that are attained at least 50% of the time by this method on this instance.

4. MDLS in practice

We now describe how to apply MDLS to three problems from the multi-objective optimization literature: the multi-objective multi-dimensional knapsack problem (MOMDKP), the bi-objective set packing problem (BOSPP) and the bi-objective orienteering problem (BOOP).

All experimental results presented in this section are available online at http://prolog.univie.ac.at/research/MDLS, as well as reusable code (ANSI C++) of the general-purpose MDLS framework used in this section.

For each problem, we compare the results obtained with MDLS to existing results from the literature. Whenever possible, we use the same CPU as the one used for the results we compare to; otherwise, we use the fastest computer we currently have access to.

4.1. Multi-objective multi-dimensional knapsack problem

4.1.1. Problem description

The knapsack problem is a traditional combinatorial optimization problem. When considering a set of items, each with a weight and a profit, and a knapsack with a given capacity, it consists in selecting items in order to maximize the sum of profits of these selected items, while keeping the sum of their weights below the capacity of the knapsack. A multi-objective version of this problem can simply be obtained by adding new profits to each item. Another extension of the knapsack problem, called the multi-dimensional knapsack problem, consists in considering not only one weight per item, but several *dimensions*, each associated to a given capacity. Then for each item a consumption for each dimension is considered, and the capacity for each dimension has to be respected. Here we consider the combination of these last two problems, i.e. the multi-objective multi-dimensional knapsack problem (MOMDKP). A mathematical model for m dimensions and k objectives, as presented by Florios et al. [19], follows:

$$\begin{aligned} &\max \quad Px \\ &\text{s.t.} \quad Wx \le c \\ &x = (x_1, \ldots, x_n)^T \in \{0,1\}^n \\ &P \in \mathbb{R}^{+(k \times n)}, \quad W \in \mathbb{R}^{+(m \times n)}, \quad c = (c_1, \ldots, c_m)^T \in \mathbb{R}^m \end{aligned} \tag{1}$$

Although it is not mandatory, it is common to have $k=m$; it is the case for all the test instances used in this paper. In the last years, the MOMDKP has been used as a standard test problem for multi-objective optimization methods. Zitzler and Thiele [58] introduce 12 benchmark instances with up to 750 items, 4 objectives and 4 dimensions, as well as comparative results for a number of evolutionary algorithms on these instances, including their SPEA. In a later work [57], they compare these results to SPEA2 and to NSGA-II, which both outperform all previous algorithms. Jaszkiewicz [28] develops a multi-objective genetic

local search (MOGLS) method, which outperforms all previous methods on the same instances. João Alves and Almeida [1] introduce a multiple objective Tchebycheff based genetic algorithm (MOTGA) for the multi-dimensional knapsack problem, and compare it to the results of Jaszkiewicz. The main conclusion is that although MOTGA produces less non-dominated solutions, they are of slightly better quality.

Some exact methods have also been developed for the MOMDKP. A natural border in terms of difficulty seems to lie between the bi- and tri-objective cases: although there are several methods to solve bi-objective instances with several hundred items, it is not possible when considering three objectives. Laumanns et al. [34] propose the first exact method for the tri-objective knapsack problem, which is an epsilon-constraint framework, and solve instances with up to 50 items in 24 h. Bazgan et al. [3] propose a dynamic programming algorithm for the multi-objective single-dimensional knapsack problem (MOKP). They solve bi-objective instances with up to 4000 items; those are all instances with correlated profits. They also solve bi-objective instances with uncorrelated profits and up to 700 items. Additionally, they solve tri-objective uncorrelated instances with up to 110 items, and conflicting instances with up to 60 items. Florios et al. [19] present a multi-criteria branch and bound (MCBB) algorithm for the MOMDKP. MCBB was introduced by Mavrotas and Diakoulaki [36]; it is based on the well-known branch and bound algorithm, but uses a set of non-dominated solutions instead of a single upper bound for fathoming nodes. This method is applied to the MOMDKP with three objectives and three dimensions, with five test instances of respectively 10, 20, 30, 40 and 50 items. Since the run time of MCBB can be very high, they also implement two metaheuristics for this problem, NSGA-II and SPEA2, and evaluate them with the Pareto sets computed with MCBB.

4.1.2. Problem-dependent MDLS components for the MOMDKP

When applying MDLS to a specific problem, one must design a way to provide starting solutions, as well as single-objective local searches. Normalization factors are computed for each objective, such that the normalized sum of scores on all items is identical for each objective. With these, we can compute a normalized sum of scores p_i for each item. A simple construction heuristic then consists in iteratively selecting unselected items with roulette wheel, using p_i as weights, until no item can be selected without violating capacity constraints. We construct two starting solutions using this simple heuristic. Additionally, for each objective, we use a greedy construction algorithm. More precisely, we use the greedy recreate operator (described further) on an empty solution, once per objective. Overall, this results in a very small starting set of solutions. For instance when working with three objectives, the cardinality of the starting set of solutions is at most 5 (only non-dominated solutions are kept).

In order to produce neighborhoods for the MOMDKP, we compute a normalized weight w_i over all dimensions for each item i. Normalization factors are computed, such that the total weighted sum of consumption over all items is the same for each dimension. For each item i we can then simply compute w_i by summing its weighted consumption over all dimensions. The neighborhoods we use are combinations of ruin and recreate operations. We consider the following ruin operations:

- *random*: consists in removing from the solution a randomly chosen item.
- $worst_k$: remove the item i which brings the smallest p_{ik}/w_i for given objective k, where p_{ik} is the profit associated to item i and objective k.
- $conflict_l$: remove the selected item with the highest consumption for dimension l.

Table 1
N_k: neighborhoods for objective k for the multi-objective multi-dimensional knapsack problem.

Neighborhood		Ruin operation	Recreate operation
1		*random*	$greedy_k$
2		$worst_k$	*random*
$2+j$	$\forall j \in [1,\ldots,K]$	$worst_j$	$greedy_k$
$2+K+l$	$\forall l \in [1,\ldots,m]$	$conflict_l$	$greedy_k$

The number of items removed from the solution is chosen randomly between 1% and 60% of the number of selected items in this solution. Additionally, we consider the following recreate operations:

- $greedy_k$: select the item i with the highest p_{ik}/w_i for given objective k.
- *random*: randomly select an item.

Now for each objective k we can define a set N_k of neighborhoods. Table 1 gives a description of N_k, assuming K objectives and m dimensions. For the MOMDKP, local search (line 6 in Algorithm 1) simply consists in ruining and recreating once the incumbent solution, i.e. it consists in one LNS iteration. For each LNS iteration the neighborhood used is selected randomly, each neighborhood being equiprobable.

Ruining a solution is always feasible, as removing elements from a solution never results in a constraint violation. For both recreate operations, only feasible moves are considered, i.e. the only items that are considered are those that can be selected without violating any constraint. This guarantees that this LNS will never, starting from a feasible solution, provide an unfeasible solution. Since the first starting solutions are provided by calling recreate operators on empty solutions (which are feasible), only feasible solutions will be considered over a whole MDLS run.

We note here that Ulungu et al. [52] also use a neighborhood operation for the MOMDKP which consists in ruining and recreating parts of a solution, in the context of their multi-objective simulated annealing (MOSA) objective. There are however two major differences. The first difference concerns the portion of the solution that is ruined. In MOSA, items are removed from the solution until each of the unselected items can be individually selected without violating any constraint. In MDLS, we use large neighborhoods which can destroy up to 60% of the solution, regardless of whether unselected items become selectable or not. In practice and on the instances on which we apply MDLS, our rule tends to ruin larger portions of the solution. The second difference concerns neighborhood exploration. In MOSA, both ruining and recreating the solution are performed randomly, and exploration is delegated to the simulated annealing algorithm. In MDLS, exploration is performed heuristically. In the case of this MDLS algorithm for the MOMDKP, exploration is guided by the combination of ruin and recreate operators. Furthermore, we note that we never use random ruin and random recreate operators together, as it is the case with MOSA.

4.1.3. Experimental results

In a first stage, we use the tri-objective instances for which exact Pareto fronts are known. There are five instances with 10, 20, 30, 40 and 50 items, for which exact results are published by Laumanns et al. [34] and by Florios et al. [19]. Results for the instance with 10 items are not available anymore, but this instance can be solved by enumeration in a few seconds (there are $2^{10}=1024$ possible solutions). We compare the results of MDLS to the exact Pareto front, as well as to two metaheuristics

also analyzed by Florios et al., NSGA-II and SPEA2; they use standard implementations for both. They compare these two methods using two indicators, the percentage of Pareto-optimal solutions found (*coverage* metric) and the average Euclidean distance to the exact set (*convergence* metric). We allow 5×10^5 iterations for MDLS. Table 2 gives the metric values for all three methods on each instance, the instance name indicating the number of items. Average, minimum and maximum values over 10 runs are reported by Florios et al.; we also provide these over 10 runs. Bold is used for best values. For both reported indicators, MDLS outperforms NSGA-II and SPEA2. Regarding CPU times, Florios et al. mention that for instance 3kp50, NSGA-II requires approximately half an hour, while SPEA2 requires 2.5 h, both on an Opteron 880 CPU. In the case of MDLS, the longest run took 5.76 s on a 2.67 GHz Intel Xeon processor.

In a second stage we use the test instances by Zitzler and Thiele, in order to compare to other metaheuristics. As mentioned earlier, the competition on these instances has been important in the last decade.

To our knowledge the two best existing heuristics at the moment for the MOMDKP are MOTGA [1] and MOEA/D [56]. Results are also available or reproducible for both methods. Both MOTGA and MOEA/D were published in the same time period, and there exists no comparison of both in the literature to our knowledge. Since both methods provide the best results available that we are aware of, we use them for assessing the quality of MDLS. In the case of MOTGA, we use the results (non-dominated sets) available online and the CPU times provided [1]. In the case of MOEA/D, we run the code provided by the authors on our computer.

There are nine instances named *n.m* where *n* is the number of items and *m* is the number of objectives and dimensions. We run MDLS 10 times with 200,000 iterations on each instance. We also run MOEA/D 10 times on each instance with the default parameter setting provided with the code available online. We consider the first 10 runs of MOTGA for each instance (there are 20 MOTGA runs per instance in total). MOTGA uses a different population size depending on the size of the problem. More precisely, this size is 20 for 250 items, 30 for 500 items and 40 for 750 items. MOEA/D uses a different parameter setting for each instance with regards to population size and aggregated objective functions. In our opinion this constitutes a bias to the comparison, in favor of MOEA/D. MDLS is set to run 200,000 iterations for each instance and has no other parameter.

We compare all three methods using the hypervolume (H) and unary multiplicative epsilon (ϵ) indicators, after normalizing all objective values to [0, 1]. The reference set for a given instance, used for normalizing and for the unary multiplicative epsilon indicator, consists of the non-dominated union of all solution sets provided by all compared methods on this given instance. Computing efficiently the hypervolume indicator is no trivial task when the number of objectives exceeds two; we use the hypervolume calculation program described in Fonseca et al. [20] and available online at http://iridia.ulb.ac.be/~manuel/hypervolume. We also provide the average cardinality of the produced approximation set, since this varies a lot between the methods. These results are summarized in Table 3. We also present average CPU times for the three algorithms per instance in Table 4. The times for MOTGA are reported for an Intel Pentium 4 CPU at 3.2 GHz. MOEA/D and MDLS are run on the same computer, which is also

Table 2
Metric values for small MOMDKP instances: NSGA-II, SPEA2 and MDLS.

Instance	Coverage			Convergence		
	NSGA-II	SPEA2	MDLS	NSGA-II	SPEA2	MDLS
3kp20						
Average	0.82	0.85	**1.0**	0.0057	0.0020	**0.0**
Min	0.72	0.79	**1.0**	0.0000	0.0000	**0.0**
Max	0.87	0.89	**1.0**	0.0160	0.0123	**0.0**
3kp30						
Average	0.773	0.803	**0.999**	0.0034	0.0026	**0.0001**
Min	0.733	0.780	**0.995**	0.0021	0.0011	**0.0**
Max	0.810	0.851	**1.0**	0.0048	0.0047	**0.0003**
3kp40						
Average	0.592	0.639	**0.930**	0.0076	0.0066	**0.0005**
Min	0.566	0.609	**0.913**	0.0056	0.0048	**0.0002**
Max	0.620	0.674	**0.938**	0.0098	0.0085	**0.0008**
3kp50						
Average	0.572	0.821	**0.853**	0.0719	0.0714	**0.0010**
Min	0.560	0.799	**0.839**	0.0711	0.0708	**0.0006**
Max	0.587	0.834	**0.871**	0.0733	0.0724	**0.0013**

Coverage: larger is better. Convergence: smaller is better.

Table 4
Average computational effort per instance, in seconds, for the multi-objective multi-dimensional knapsack problem.

Instance	MOTGA	MOEA/D	MDLS
250.2	1.5	3.44	7.77
250.3	2.7	7.45	11.78
250.4	4.2	27.98	20.19
500.2	7.2	10.38	14.17
500.3	12.8	24.15	19.53
500.4	18.2	97.21	38.20
750.2	19.5	21.21	20.56
750.3	33.4	45.36	28.63
750.4	51.9	245.60	54.23

Table 3
Hypervolume (H), unary epsilon (ϵ) and non-dominated front size ($|front|$) indicator values for the MOMDKP (average values over 10 runs).

Instance	MOTGA			MOEA/D			MDLS		
	H	ϵ	$\|front\|$	H	ϵ	$\|front\|$	H	ϵ	$\|front\|$
250.2	0.80312	1.0151	99.5	0.79597	1.0237	141.7	**0.80706**	**1.0123**	216.0
250.3	0.53296	1.0638	536.6	0.53832	1.0751	1868.9	**0.55118**	**1.0369**	3842.9
250.4	0.32249	1.0958	1074.3	**0.34433**	**1.0868**	7669.4	0.33838	1.0924	16,410.8
500.2	**0.78860**	**1.0070**	168.1	0.75833	1.0451	174.9	0.78579	1.0096	341.2
500.3	**0.51645**	1.0428	1073.8	0.49010	1.0722	2702.8	0.50982	**1.0379**	5093.8
500.4	0.30908	1.0713	2092.3	0.28516	1.1226	10110.3	**0.30934**	**1.0615**	22,039.2
750.2	**0.77636**	**1.0040**	252.2	0.73043	1.0538	190.8	0.76716	1.0142	450.4
750.3	**0.52101**	1.0396	1425.0	0.45964	1.0926	2449.7	0.50519	**1.0327**	5838.0
750.4	**0.29956**	1.0632	3076.6	0.24082	1.1416	11343.6	0.28415	**1.0562**	26,868.0

H: larger is better. ϵ: smaller is better.

an Intel Pentium 4 CPU at 3.2 GHz. Looking at these results, we can see that MDLS is competitive with both other methods in terms of solution quality. None of the three methods seems to dominate the other two. We also note that it never happens that MDLS provides the worst indicator value, be it for hypervolume or unary epsilon.

Having examined various experiments, we conclude that MDLS is competitive with existing multi-objective metaheuristics for the MOMDKP. This conclusion should be understood in the context of an important competition on metaheuristics for this specific problem. With a CPU effort comparable to that required by state-of-the-art alternatives (MOTGA and MOEA/D), results of competitive quality are provided.

In order to provide insight on the neighborhoods and the way they work within MDNS, we provide a simple analysis. We define the *success rate* of a neighborhood as the number of times it provides a new non-dominated solution divided by the number of times it is called. In Appendix A.1 we indicate the success rate of each neighborhood used for the MOMDKP. Tables A1–A3 contain these values for the MDLS runs on problems with respectively 2, 3 and 4 objectives. Interestingly, the most successful neighborhoods are those that greedily insert items following objective i after greedily removing items following any objective other than i. We also note that more objectives involve more successful moves, which is likely due to the fact that more objectives imply more solutions in the Pareto front. Finally, we also note that no neighborhood is useless. The success rates vary, but within a given table the factor between the lowest and highest success rates is around 10. Without surprise the neighborhoods that use the *random* recreate operator are among the least successful, although they still provide some improvement.

4.2. Bi-objective set packing problem

4.2.1. Problem description

In the set packing problem we consider a set I of elements and a set J of m subsets of I. Each subset has a score c_j, and the goal is to maximize the total score of selected subsets, while each element of I may only appear in at most one selected subset. The set packing problem has received less attention than the covering and partitioning variants. One application is the maximization of profit when bidding on groups (subsets) of items [16], in which case it is not possible to bid several times on the same item. Another application is that of planning railway sections in which only one train may be present at a time [35].

To our knowledge the bi-objective set packing problem (BOSPP) was first mentioned by Delorme [13]. Two profits are considered for each subset, one per objective, and the goal is to maximize total profit for both objectives. Considering a set I of n items and a set J of m subsets, the authors propose the following formulation:

$$\begin{aligned}
&\max \quad z^1 = \sum_{j \in J} c_j^1 x_j \\
&\max \quad z^2 = \sum_{j \in J} c_j^2 x_j \\
&\text{s.t.} \quad \sum_{j \in J} t_{ij} x_j \leq 1 \quad \forall i \in I \\
&x_j \in \{0,1\} \quad \forall j \in J
\end{aligned} \tag{2}$$

where x_j is the binary variable associated to the selection of set j, c_j^1 and c_j^2 are the two profits for subset j, and t_{ij} indicates whether item i is covered by subset j or not. They then provide three metaheuristics for this problem. The first one is an Aggressive SPEA, which the authors introduce in the paper. Among its characteristics are four differences with SPEA, including removing the clustering procedure, storing all non-dominated solutions, the application of local search when a new solution is created and also at the end as post-processing. Their second metaheuristic is a λ-GRASP. It embeds a standard GRASP procedure (see e.g. [17]) for single-objective optimization. This procedure is iterated independently with various weighted sum objective functions. The third method proposed by the authors is a hybrid of the A-SPEA and the λ-GRASP.

Set packing problems can also be seen as special cases of knapsack problems. However, as mentioned in Delorme et al. [14], this property does not help in finding good solutions. Indeed, we tried to apply our MDLS for the MOMDKP from the previous section, with very poor results.

4.2.2. Problem-dependent MDLS components for the BOSPP

In order to provide a starting set of solutions, we use the same method as the one described in Section 4.1 for the multi-dimensional knapsack problem.

We now introduce the concept of *conflict*, used in some of the neighborhoods for the BOSPP. Two subsets are conflicting if they have at least one element in common. We can construct a graph $G=\{V,E\}$ where the nodes in G correspond to the subsets, and for each conflict there is an edge in E. A feasible solution to the set packing problem is then associated to a stable set in this graph, and the maximum stable set problem is a special case of the single-objective set packing problem, where all subsets have the same profit.

We consider the following ruin operations:

- *random*: consists in removing from the solution a randomly chosen subset.
- *worst*$_k$: remove the subset which brings the smallest profit when considering a given objective k.
- *conflict_random*: randomly select an unselected subset s, then deselect from the solution all subsets that have a conflict with s.
- *conflict_greedy*: select the unselected subset s with least conflicting selected subsets in the solution; then deselect from the solution all subsets that have a conflict with s.
- *conflict_alt*: removes the selected subset with the highest number of conflicts.

The ruin quantity, i.e. the number of subsets removed from the solution, is a number chosen randomly between 1% and 60% of the number of selected subsets in this solution. Additionally, we consider the following recreate operations:

- *greedy*$_k$: select the subset which brings the highest profit for a given objective k.
- *random*: randomly select a subset.
- *closer*: let U be the set of subsets that cannot be selected because they have a conflict with at least one selected subset; this operator selects the subset that minimizes $|U|$.

Like for the MOMDKP, only feasible solutions are considered: ruining a solution is always feasible, and the recreate operations only consider selecting subsets if their selection does not result in a constraint violation.

Now for each objective k we can define a set N_k of neighborhoods. Table 5 gives a description of N_k. In the case of BOSPP, local search (line 6 in Algorithm 1) consists in iterating 10 times the following steps: (i) ruin the incumbent solution, (ii) recreate it and (iii) set the resulting neighbor as new incumbent. The same neighborhood is used for all 10 steps; as with the MOMDKP, the neighborhood is selected randomly, with an equal probability for each neighborhood. We let MDLS run for 50,000 iterations.

Table 5
N_k: neighborhoods for objective k for the bi-objective set packing problem.

Neighborhood	Ruin operation	Recreate operation
1	*random*	$greedy_k$
2	$worst_k$	$greedy_k$
3	*conflict_greedy*	$greedy_k$
4	*conflict_random*	$greedy_k$
5	*conflict_alt*	$greedy_k$
6	$worst_k$	*closer*
7	$worst_k$	*random*

Table 6
Indicator values for the BOSPP.

Indicator	A-SPEA	λ-GRASP	Hybrid	MDLS
Coverage	77.67	80.22	95.90	**96.63**
Convergence	3.60	4.28	0.70	**0.65**
H	98.90	99.83	99.88	**99.96**

Coverage: larger is better. Convergence: smaller is better. H: larger is better.

Table 7
Average computational effort, in seconds, for the bi-objective set packing problem.

m	Delorme et al. (A-SPEA, λ-GRASP and hybrid) Intel Pentium III 800 MHz	MDLS Intel Xeon 2.67 GHz
100	50	16.38
200	200	28.05

4.2.3. Experimental results

To our knowledge, the only available experimental data for the BOSPP is contained in the paper by Delorme et al. [14]. The test problems consist of 120 instances with 100 or 200 variables and from 300 to 1000 constraints. Since exact Pareto sets are also known, they can evaluate precisely the quality of each method. The three indicators used are the percentage of Pareto-optimal solutions found (M1 or *coverage*), the Euclidean distance to the Pareto set (*convergence*), and the hypervolume (H). More precisely, they report the percentage of the hypervolume of the Pareto set. Delorme et al. run each of their algorithms 16 times, and report average values on all three indicators for each of their methods. We report average values over 10 runs of MDLS for each of these indicators in Table 6. Best values over all methods are boldfaced.

On average, the hybrid method clearly dominates A-SPEA and λ-GRASP, but the MDLS is even better. However, this comes at the cost of a higher CPU time. Delorme et al. determine allowed CPU time based on the instance. More precisely, they compute a reference time $T_{REF} = 10^{-3}$ m^2 s (m being the number of subsets). They then provide results for total CPU times of value $k \cdot T_{REF}$, where k is 1, 2, 3, 4 and 5. Since the results for $k=5$ are always better than the others, we only report these here. We also report the average CPU time used by MDLS. These values are reported in Table 7. Although the values are lower for MDLS, it is clear that the CPU we use is also faster, clock speed aside. Therefore it is hard to conclude anything beside the fact that both approaches require CPU time of the same order of magnitude; MDLS is probably slower though. Additionally, the difference in CPU effort between 100- and 200-subset classes is much smaller for MDLS. However the stopping conditions are so different in both methods that it is not possible to conclude anything regarding scalability. In the end, it is safe to conclude that MDLS provides good quality results within reasonable CPU time.

We provide insight on the neighborhoods and their success rates in Appendix A.2. Table A4 indicates the success rates for each neighborhood. A first observation is that the success rates are very low. However Table 6 clearly indicates that most of the Pareto optimal solutions are systematically found. In fact the more Pareto optimal solutions have been identified during a run, the least likely any neighborhood is to provide a new Pareto optimal solution during this same run. The average size of the Pareto front on all BOSPP instances is 21.1583, while the largest one is 126. Once these all have been identified, success becomes impossible. We also note that the factor between the lowest and highest success rates is less than 3.

4.3. Bi-objective orienteering problem

4.3.1. Problem description

The bi-objective orienteering problem (BOOP) was introduced by Schilde et al. [46]; it is similar to the bi-objective traveling salesman problem with profits solved by Bérubé et al. [4]. Quoting Schilde et al., "The motivation of the problem stems from planning individual tourist routes in cities and rural areas". The different objectives correspond to various centers of interest for the tourists.

The BOOP is a bi-objective extension of the well-known orienteering problem (OP), introduced by Tsiligirides [51]. In the OP, a set of points is given along with travel durations between those points. Starting and ending points are specified, while all other points are called *control points*. To each control point is associated a score. The goal is then to maximize the total score by visiting these control points, while keeping the total travel time under a given limit. Tsiligirides proposed benchmark instances; these were completed by other instances from Chao [6]. All these instances, containing up to 66 points, have been used in several research works since then (see e.g. [7,46,50]), and are no longer considered difficult. Schilde et al. introduced bi-objective instances by adding a second score to each control point of the existing instances. They also introduced new, bigger bi-objective instances with up to 2143 control points. These new instances are based on tourism attractions and use road network information for the distance matrix. These two scores can be seen as different centers of interest for tourists, and should both be maximized.

Bérubé et al. provide an exact method, and use it to solve different instances with up to 150 control points. In the following, we focus on the heuristic solution of large instances.

Schilde et al. provide a mathematical model for the BOOP, considering profit s_{ik} for point i and objective k, and travel duration c_{ij} between points i and j. Their model is based on a directed graph $G=\{V,A\}$ where nodes v_0 and v_{n+1} are the mandatory starting and ending points. It is a generic model for any number of objectives:

$$\max \quad f_k(y) = \sum_{v_i \in V\setminus\{v_0,v_{n+1}\}} s_{ik} y_i \quad (k = 1 \dots K) \tag{3}$$

s.t.

$$\sum_{v_j \in V\setminus\{v_i\}} x_{ij} = y_i \quad (v_i \in V\setminus\{v_{n+1}\}) \tag{4}$$

$$\sum_{v_i \in V\setminus\{v_j\}} x_{ij} = y_j \quad (v_j \in V\setminus\{v_0\}) \tag{5}$$

$$\sum_{\{v_i,v_j\} \in S} x_{ij} \le |S| - 1 \quad (S \subseteq V \wedge S \neq \emptyset) \tag{6}$$

$$y_0 = y_{n+1} = 1 \tag{7}$$

$$\sum_{(v_i,v_j) \in A} c_{ij} x_{ij} \le T_{max} \tag{8}$$

$$x_{ij} \in \{0,1\} \quad ((v_i,v_j) \in A) \tag{9}$$

$$y_i \in \{0,1\} \quad (v_i \in V) \tag{10}$$

Constraints (4) and (5) ensure that for each visited control point there is an ingoing and an outgoing arc; Constraints (6) are for subtour elimination; Constraint (7) makes the tour start and end at the correct points, and Constraint (8) limits the total tour duration.

4.3.2. Problem-dependent MDLS components for the BOOP

In order to provide a starting set of solutions, we use the same method as the one described in Section 4.1 for the multi-dimensional knapsack problem. A problem-specific feature of the BOOP requires to define a specific dominance rule: for a given set of k control points, there exists up to $k!$ feasible tours, i.e. different solutions, all with the same objective vector. Therefore the dominance rule is modified as follows: if two tours visit the same set of control points, the one with the shorter duration dominates the other one.

We consider the following ruin operations:

- *worst*$_k$: remove the control points which bring the smallest profit when considering a given objective k.
- *related*: remove control points that are related. What we call related control points here is a set of control points that form a sequence in the solution. The position of this sequence in the solution is chosen randomly, depending on the quantity of nodes to remove.
- *conflict*: a control point yields a conflict with its predecessor and successor when visiting this point takes a long time. For each control point in a solution we can compute how much time would be gained by not visiting this point anymore; then the points with the highest conflict value are removed.
- *random*: consists in removing from the solution control points that are randomly selected.

The ruin quantity, i.e. the number of control points removed from the solution, is a number selected randomly between 1% and 50% of the number of points in this solution. Additionally, we consider the following recreate operations:

- *greedy*$_k$: insert the control points bringing the highest profit for a given objective k.
- *closer*: insert the control points that are close to the current solution, i.e. those that would induce the smallest extra traveling time.
- *random*: insert randomly selected control points in the solution.

Control points are inserted as long as it is possible without violating the route duration constraint. This guarantees that no constraint is violated in the solutions produced by these operators, similarly to the operators for the MOMDKP and the BOSPP. Each of these recreate operations is applied in a cheapest insertion fashion: the position where a node is inserted in the tour is systematically the one that brings the smallest extra traveling time. Additionally, a noise is applied to heuristic criteria in order to provide diversification. This is performed by multiplying the heuristic evaluation of moves by a random number between 0.75 and 1.25.

Using all previously described operators, a set N_k of neighborhoods is constructed for each objective k. Table 8 gives a description of N_k. For the BOOP, local search (line 6 in Algorithm 1) simply consists in ruining and recreating once the incumbent solution. Again, the neighborhood is selected randomly, all neighborhoods being equiprobable. A problem-specific diversification mechanism is also used. It consists in, given a solution x', looking for solutions that do not contain any of the control points that are visited by x'. This can be seen as another large neighborhood around x'. In order to explore quickly this very large neighborhood, heuristics are used. More precisely, once a solution x' has been produced through the ruin and recreate steps, a new solution is produced by forbidding all the points in x' and recreating using the same operator that was used to produce x'. This diversification step is executed once every 30 calls to a same single-objective local search. The total number of iterations of MDLS is set to 50,000.

Table 8
N_k: neighborhoods for objective k for the bi-objective orienteering problem.

Neighborhood	Ruin operation	Recreate operation
1	*random*	*greedy*$_k$
2	*worst*$_k$	*greedy*$_k$
3	*related*	*greedy*$_k$
4	*conflict*	*greedy*$_k$
5	*worst*$_k$	*closer*
6	*worst*$_k$	*random*

Table 9
Indicator values for the bi-objective orienteering problem: 20% attainment sets.

Instances	VNS		ACO		MDLS	
	H	ϵ	H	ϵ	H	ϵ
p21	0.374	1.123	0.374	1.123	**0.718**	**1.000**
p32	0.437	1.035	0.446	1.035	**0.542**	**1.000**
p33	0.586	1.094	0.587	1.090	**0.772**	**1.000**
dia	0.654	1.014	0.638	1.021	**0.676**	**1.000**
squ	0.656	1.008	0.649	1.014	**0.673**	**1.001**
pad	0.497	**1.000**	0.491	**1.000**	**0.501**	**1.000**
wie	0.571	1.068	0.543	1.059	**0.710**	**1.006**
ktn	0.619	1.052	0.645	1.041	**0.724**	**1.014**
stm	0.676	1.029	0.660	1.033	**0.749**	**1.003**
noe	0.563	1.073	0.574	1.086	**0.825**	**1.021**

H: larger is better. ϵ: smaller is better.

4.3.3. Experimental results

In order to solve the BOOP, Schilde et al. [46] provide two methods: a Pareto variable neighborhood search (P-VNS) and a Pareto ant colony optimization (P-ACO). They consider six indicators that they compute for 20%, 50% and 80% attainment sets. For the sake of simplicity, we only report two of these indicators, hypervolume and unary epsilon. No Pareto front is available for these instances, and we found new non-dominated solutions that were unknown before. Therefore we provide new normalization functions which take into account these new non-dominated solutions. Since we have access to all solutions from Schilde et al., we also normalize their solutions and recompute the indicator values for their VNS and ACO.

The test instances are categorized, and each category represents problems with the same distance matrix but different time limits. Tables 9–11 summarize the results of all three methods on both hypervolume (H) and unary epsilon (ϵ) indicators, respectively for 20%, 50% and 80% attainment sets, with one line giving average indicator values for a category of instances. Since no exact method exists for this problem, the Pareto set is not known; therefore a heuristic reference set is used for the computation of the unary epsilon indicator. This heuristic front is obtained by considering all non-dominated solutions found by all three methods (P-VNS, P-ACO, MDLS).

The MDLS outperforms both previous methods, with the notable exception of the unary epsilon indicator for instance stm and 80% attainment. On the same instance category the

Table 10
Indicator values for the bi-objective orienteering problem: 50% attainment sets.

Instances	VNS		ACO		MDLS	
	H	ϵ	H	ϵ	H	ϵ
p21	0.374	1.123	0.362	1.130	**0.718**	**1.000**
p32	0.433	1.035	0.446	1.035	**0.531**	**1.006**
p33	0.583	1.085	0.584	1.090	**0.770**	**1.000**
dia	0.642	1.016	0.623	1.029	**0.673**	**1.003**
squ	0.642	1.016	0.638	1.018	**0.669**	**1.002**
pad	0.497	**1.000**	0.491	**1.000**	**0.501**	**1.000**
wie	0.534	1.088	0.522	1.063	**0.690**	**1.016**
ktn	0.589	1.068	0.601	1.063	**0.683**	**1.032**
stm	0.649	1.036	0.624	1.050	**0.729**	**1.015**
noe	0.494	1.104	0.536	1.106	**0.761**	**1.043**

H: larger is better. ϵ: smaller is better.

Table 11
Indicator values for the bi-objective orienteering problem: 80% attainment sets.

Instances	VNS		ACO		MDLS	
	H	ϵ	H	ϵ	H	ϵ
p21	0.367	1.130	0.360	1.130	**0.718**	**1.000**
p32	0.412	1.036	0.440	1.035	**0.529**	**1.006**
p33	0.576	1.075	0.583	1.085	**0.760**	**1.006**
dia	0.626	1.024	0.610	1.034	**0.670**	**1.005**
squ	0.621	1.025	0.629	1.021	**0.663**	**1.006**
pad	0.497	**1.000**	0.489	1.004	**0.501**	**1.000**
wie	0.488	1.107	0.501	1.073	**0.654**	**1.033**
ktn	0.553	1.078	0.565	1.078	**0.613**	**1.067**
stm	0.611	**1.049**	0.595	1.069	**0.634**	1.060
noe	0.420	1.141	0.510	1.121	**0.662**	**1.087**

H: larger is better. ϵ: smaller is better.

Table 12
Average computational effort, in seconds, for the bi-objective orienteering problem.

Instances	ACO and VNS	MDLS
p21	1.35	1.18
p32	3.25	1.84
p33	5.14	2.02
dia	14.97	5.20
squ	16.17	4.95
pad	1.17	0.64
wie	31.48	19.90
ktn	84.53	24.99
stm	36.57	20.08
noe	3319.52	193.81

hypervolume indicator is better for the MDLS though. The unary epsilon indicator grades the quality of the worst solution found. Comparison data with regards to CPU effort is provided in Table 12. Each line corresponds to a class of instances, and the average run time is reported for each method, as well as the type of computer used. Since Schilde et al. used the exact same CPU time for both their methods, only one is reported. Times are reported in seconds. We use the same computer on which Schilde et al. run ACO and VNS. We simply conclude that MDLS provides better results for the BOOP, within comparable computational times. The *noe* instance set is the largest of all (2143 control points), which explains why all methods need more CPU time for that instance than for any other. Because of the travel time limitation and depending on its value, some control points can never be reached. Therefore it is possible to discard those control points before starting the optimization. Such a pre-processing is performed before MDLS, but not before ACO and VNS. In the case of *noe*, this pre-processing has a major impact, which explains the difference in CPU times between ACO and VNS on one side and MDLS on the other side.

We provide insight on the neighborhoods and their success rates in Appendix A.3. Table A5 indicates the success rates for each neighborhood. The factor between the lowest and highest success rates is less than 2. Again, the neighborhoods incorporating a random component are among the least successful.

5. Summary

In this paper, we introduced multi-directional neighborhood search (MDLS), a new metaheuristic for multi-objective optimization. MDLS is based on the principle of using single-objective local search to iteratively improve the non-dominated front. An important feature is that only non-dominated solutions are kept. Another feature is that once problem-dependent components (i.e. single-objective local search) are defined, the only parameter left is CPU budget. We provided a general framework for MDLS, and showed how to apply it to three different problems, every time with competitive results on benchmark data. These three problems were selected for the simplicity of their formulation, and the availability of benchmark data and/or Pareto sets. For all three problems, the objectives considered were uncorrelated. On all standard benchmark instances for the multi-dimensional multi-objective knapsack problem, bi-objective set packing problem and bi-objective orienteering problem, MDLS produce results competitive with the best known solution method known so far, with a required CPU effort of the same order of magnitude. Although two of these problems (the BOOP and the BOSPP) have direct applications, all solved problems have a fairly simple formulation.

One important feature of MDLS is that it relies on single-objective local search. This has the advantage that good single-objective local search methods are known for a number of problems. However it can also be seen as a disadvantage: it means that efficient single-objective local search has to be designed and implemented for each objective in the problem at hand. If the problem structure is relatively simple, like it is the case with all three problems studied in this paper, then it does not really matter, as the local searches are very similar for all considered objectives. However if the objectives are very different, more work will be required to design and implement an efficient MDLS. This is a disadvantage of MDLS to most other methods, since they usually do not rely on single-objective local search.

The next step is to validate MDLS on applied problems with complicated real-world constraints. Such problems already exist in the literature, like for instance the capacitated vehicle routing

Table A1
Success rate of each neighborhood for the MOMDKP; instances with two objectives.

Ruin operation	Recreate operation	# Used	# Success	% Success
$conflict_1$	$greedy_0$	1,332,815	1590	0.1193
$conflict_1$	$greedy_1$	1,333,919	1630	0.1222
$worst_1$	*random*	1,333,385	1136	0.0852
$worst_1$	$greedy_0$	1,332,403	10,916	0.8193
$worst_1$	$greedy_1$	1,334,305	6733	0.5046
$conflict_0$	$greedy_1$	1,331,771	1299	0.0975
$conflict_0$	$greedy_0$	1,334,930	2114	0.1584
$worst_0$	$greedy_1$	1,333,506	11,834	0.8874
$worst_0$	$greedy_0$	1,332,752	5670	0.4254
$worst_0$	*random*	1,333,948	1215	0.0911
random	$greedy_1$	1,333,154	4602	0.3452
random	$greedy_0$	1,333,192	4829	0.3622

Table A2
Success rate of each neighborhood for the MOMDKP; instances with three objectives.

Ruin operation	Recreate operation	# Used	# Success	% Success
$conflict_1$	$greedy_0$	997,439	7307	0.7326
$conflict_1$	$greedy_1$	1,000,465	7371	0.7368
$conflict_1$	$greedy_2$	999,335	5861	0.5865
$worst_0$	$greedy_2$	1,000,609	50,618	5.0587
$worst_0$	$greedy_1$	1,000,371	51,216	5.1197
$worst_0$	$greedy_0$	1,004,744	26,696	2.6570
$worst_0$	*random*	1,005,449	5464	0.5434
random	$greedy_1$	999,387	14,401	1.4410
random	$greedy_2$	1,000,222	15,492	1.5489
$conflict_2$	$greedy_2$	998,808	12,295	1.2310
$conflict_2$	$greedy_1$	1,000,780	6432	0.6427
$worst_2$	$greedy_0$	998,374	41,695	4.1763
$worst_2$	*random*	1,001,561	5283	0.5275
$worst_2$	$greedy_2$	998,316	23,455	2.3495
$worst_2$	$greedy_1$	998,969	48,670	4.8720
$worst_1$	*random*	1,000,686	5995	0.5991
$worst_1$	$greedy_0$	1,000,264	39,380	3.9370
$worst_1$	$greedy_1$	999,195	35,775	3.5804
$worst_1$	$greedy_2$	1,000,834	49,384	4.9343
$conflict_0$	$greedy_2$	1,000,355	11,436	1.1432
$conflict_0$	$greedy_1$	1,000,187	10,451	1.0449
$conflict_0$	$greedy_0$	997,506	8008	0.8028
$conflict_2$	$greedy_0$	997,247	8089	0.8111
random	$greedy_0$	999,017	13,350	1.3363

Table A3
Success rate of each neighborhood for the MOMDKP; instances with four objectives.

Ruin operation	Recreate operation	# Used	# Success	% Success
$conflict_1$	$greedy_3$	799,992	14,724	1.8405
$conflict_1$	$greedy_0$	799,018	17,937	2.2449
$conflict_1$	$greedy_1$	799,637	13,317	1.6654
$conflict_1$	$greedy_2$	800,281	13,727	1.7153
$worst_0$	*random*	800,797	11,747	1.4669
$conflict_3$	$greedy_3$	799,542	18,052	2.2578
$worst_0$	$greedy_3$	799,630	99,000	12.3807
$conflict_3$	$greedy_1$	801,612	17,651	2.2019
$worst_0$	$greedy_1$	800,364	82,229	10.2740
$worst_0$	$greedy_0$	800,115	59,503	7.4368
$conflict_3$	$greedy_0$	799,898	15,942	1.9930
$conflict_2$	$greedy_3$	800,911	33,444	4.1757
random	$greedy_3$	799,840	25,757	3.2203
random	$greedy_1$	799,368	25,561	3.1977
random	$greedy_2$	799,716	26,242	3.2814
$conflict_2$	$greedy_2$	800,968	17,104	2.1354
$conflict_2$	$greedy_1$	800,159	27,090	3.3856
$worst_2$	$greedy_0$	800,477	93,409	11.6692
$worst_2$	*random*	800,378	13,875	1.7336
$worst_2$	$greedy_2$	799,733	83,310	10.4172
$worst_2$	$greedy_1$	800,064	96,696	12.0860
$worst_2$	$greedy_3$	799,162	96,211	12.0390
$worst_1$	*random*	799,760	12,165	1.5211
$worst_1$	$greedy_0$	799,458	87,849	10.9886
$worst_1$	$greedy_1$	800,003	43,644	5.4555
$worst_1$	$greedy_2$	798,259	91,446	11.4557
$worst_1$	$greedy_3$	799,634	72,703	9.0920
$conflict_0$	$greedy_2$	800,570	14,331	1.7901
$conflict_0$	$greedy_1$	799,198	18,521	2.3174
$conflict_0$	$greedy_0$	800,439	15,594	1.9482
$conflict_0$	$greedy_3$	799,911	13,918	1.7399
$worst_3$	$greedy_1$	799,875	96,161	12.0220
$worst_3$	$greedy_2$	800,827	90,162	11.2586
$conflict_2$	$greedy_0$	800,151	17,661	2.2072
$worst_3$	$greedy_0$	799,488	91,175	11.4042
$worst_3$	$greedy_3$	801,018	65,091	8.1260
random	$greedy_0$	800,199	28,260	3.5316
$worst_3$	*random*	800,400	12,764	1.5947
$worst_0$	$greedy_2$	800,401	102,283	12.7790
$conflict_3$	$greedy_2$	798,907	21,367	2.6745

problem with route balancing, or the dial-a-ride problem. They will constitute the subject of our future research in this domain.

Acknowledgments

The author wishes to thank Fabien Lehuédé, Renaud Masson and Olivier Péton for fruitful discussions during which the idea of MDLS was born, Michael Schilde for providing detailed data and valuable information, Karl Doerner for his precious advices and useful comments, and Sophie Parragh for her support, help and remarkable patience. Financial support from the Austrian Science Fund (FWF Grants P20342-N13 and L628-N15) and from the Austrian Research Promotion Agency (FFG Grant 822739) is gratefully acknowledged. We also wish to thank the three anonymous referees for their fruitful comments; they helped us greatly improve the quality of this paper.

Appendix A. Comparison of the efficiency of different neighborhoods

A.1. Neighborhoods for the MOMDKP

Neighborhoods for the MOMDKP are given in Tables A1–A3.

A.2. Neighborhoods for the BOSPP

Neighborhoods for the BOSPP are given in Table A4.

A.3. Neighborhoods for the BOOP

Neighborhoods for the BOOP are given in Table A5.

Table A4
Success rate of each neighborhood for the BOSPP.

Ruin operation	Recreate operation	# Used	# Success	% Success
conflict_alt	$greedy_0$	85,742,100	13,199	0.0154
conflict_alt	$greedy_1$	85,721,520	12,712	0.0148
$worst_1$	$greedy_0$	85,720,330	8633	0.0101
$worst_1$	*closer*	85,728,840	10,668	0.0124
$worst_1$	*random*	85,678,800	6354	0.0074
conflict_greedy	$greedy_1$	85,703,270	6791	0.0079
conflict_greedy	$greedy_0$	85,730,340	7113	0.0083
$worst_0$	*random*	85,706,810	7815	0.0091
$worst_0$	*closer*	85,695,980	12,482	0.0146
$worst_0$	$greedy_0$	85,673,960	5474	0.0064
conflict_random	$greedy_0$	85,766,720	5541	0.0065
conflict_random	$greedy_1$	85,746,170	5170	0.0060
random	$greedy_0$	85,696,090	7915	0.0092
random	$greedy_1$	85,713,070	7348	0.0086

Table A5
Success rate of each neighborhood for the BOOP.

Ruin operation	Recreate operation	# Used	# Success	% Success
related	$greedy_1$	18,600,689	37,920	0.2039
related	$greedy_0$	18,594,560	38,428	0.2067
$worst_1$	$greedy_1$	18,600,299	58,577	0.3149
$worst_1$	*closer*	18,597,996	37,652	0.2025
$worst_1$	*random*	18,602,020	32,620	0.1754
$worst_0$	*random*	18,592,118	29,875	0.1607
$worst_0$	*closer*	18,602,724	34,113	0.1834
$worst_0$	$greedy_0$	18,597,332	56,447	0.3035
conflict	$greedy_0$	18,609,080	36,067	0.1938
conflict	$greedy_1$	18,601,938	41,012	0.2205
random	$greedy_0$	18,607,066	34,140	0.1835
random	$greedy_1$	18,599,938	35,048	0.1884

References

[1] João Alves M, Almeida M. MOTGA: a multiobjective Tchebycheff based genetic algorithm for the multidimensional knapsack problem. Computers & Operations Research 2007;34:3458–70.

[2] Alves MJ, Climaco J. An interactive method for 0-1 multiobjective problems using simulated annealing and tabu search. Journal of Heuristics 2000;6: 385–403.

[3] Bazgan C, Hugot H, Vanderpooten D. Solving efficiently the 0-1 multi-objective knapsack problem. Computers & Operations Research 2009;36: 260–79.

[4] Bérubé JF, Gendreau M, Potvin JY. An exact epsilon-constraint method for bi-objective combinatorial optimization problems: application to the traveling salesman problem with profits. European Journal of Operational Research 2009;194:39–50.

[5] Branke J, Greco S, Slowínski R, Zielniewicz P. Interactive evolutionary multi-objective optimization using robust ordinal regression. In: Ehrgott M, et al., editors. EMO 2009. Berlin: Springer; 2009. p. 554–68.

[6] Chao IM. Algorithms and solutions to multi-level vehicle routing problems. PhD thesis. College Park, MD, USA. Chairman-Bruce L. Golden; 1993.

[7] Chao IM, Golden BL, Wasil EA. A fast and effective heuristic for the orienteering problem. European Journal of Operational Research 1996;88: 475–89.

[8] Coello CAC, Christiansen AD. Two new GA-based methods for multiobjective optimization. Civil Engineering and Environmental Systems 1998;15:207–43.

[9] Coello Coello A, Van Veldhuizen D, Lamont G. Evolutionary algorithms for solving multi-objective problems. Kluwer Academic Publishers; 2002.

[10] Czyzak P, Jaszkiewicz A. Pareto simulated annealing—a metaheuristic technique for multiple-objective combinatorial optimization. Journal of Multi-Criteria Decision Analysis 1998;7:34–47.

[11] Da Fonseca VG, Fonseca CM, Hall AO. Inferential performance assessment of stochastic optimisers and the attainment function. In: First international conference on evolutionary multi-criterion optimization. Springer-Verlag; 2001. p. 213–25.

[12] Deb K, Pratap A, Agarwal S, Meyarivan T. A fast and elitist multi-objective genetic algorithm: NSGA II. IEEE Transactions on Evolutionary Computation 2002;6:182–97.

[13] Delorme X. Modélisation et résolution de problèmes liés à l'exploitation d'infrastructures ferroviaires. PhD thesis. Université de Valenciennes et du Hainaut Cambrésis [in French]; 2003.

[14] Delorme X, Gandibleux X, Degoutin F. Evolutionary, constructive and hybrid procedures for the bi-objective set packing problem. European Journal of Operational Research 2010;204:206–17.

[15] Doerner KF, Gutjahr WJ, Hartl RF, Strauss C, Stummer C. Pareto ant colony optimization: a metaheuristic approach to multiobjective portfolio selection. Annals of Operations Research 2004;131:79–99.

[16] Escudero L, Landete M, Marín A. A branch-and-cut algorithm for the winner determination problem. Decision Support Systems 2009;46:649–59.

[17] Festa P, Resende M. GRASP: an annotated bibliography. In: Ribeiro C, Hansen P, editors. Essays and surveys in metaheuristics. Boston: Kluwer Academic Publishers; 2004. p. 325–67.

[18] Figueira JR, Greco S, Slowínski R. Building a set of additive value functions representing a reference preorder and intensities of preference: GRIP method. European Journal of Operational Research 2009;195:460–86.

[19] Florios K, Mavrotas G, Diakoulaki D. Solving multiobjective, multiconstraint knapsack problems using mathematical programming and evolutionary algorithms. European Journal of Operational Research 2010;203:14–21.

[20] Fonseca C, Paquete L, López-Ibáñez M. An improved dimension-sweep algorithm for the hypervolume indicator. In: IEEE congress on evolutionary computation; 2006. p. 1157–63.

[21] Fonseca CM, Fleming PJ. Genetic algorithms for multiobjective optimization: formulation, discussion and generalization. In: Genetic algorithms: proceedings of the fifth international conference; 1993. p. 416–23.

[22] Framinan J, Leisten R. A multi-objective iterated greedy search for flowshop scheduling with makespan and flowtime criteria. OR Spectrum 2008;30: 787–804.

[23] Gandibleux X, Fréville A. Tabu search based procedure for solving the 0-1 multiobjective knapsack problem: the two objectives case. Journal of Heuristics 2000;6:361–83.

[24] Gandibleux X, Mezdaoui N, Fréville A. A tabu search procedure to solve multiobjective combinatorial optimization problems. In: Caballero R, Ruiz F, Steuer R, editors. Advances in multiple objective and goal programming. Heidelberg-Berlin: Springer; 1997. p. 103–8.

[25] Glover F, Laguna M. Tabu search. Norwell: Kluwer; 1997.

[26] Hapke M, Jaszkiewicz A, Slowinski R. Pareto simulated annealing for fuzzy multi-objective combinatorial optimization. Journal of Heuristics 2000;6: 329–45.

[27] Hoos H, Stützle T. Stochastic local search: foundations & applications. San Francisco, CA, USA: Elsevier/Morgan Kaufmann; 2004.

[28] Jaszkiewicz A. On the performance of multiple-objective genetic local search on the 0/1 knapsack problem—a comparative experiment. IEEE Transactions on Evolutionary Computation 2002;6:402–12.

[29] Jaszkiewicz A, Branke J. Interactive multiobjective evolutionary algorithms. In: Branke J, et al., editors. Multiobjective optimization. Berlin: Springer; 2008. p. 179–93.

[30] Jaszkiewicz A, Ferhat AB. Solving multiple criteria problems by interactive trichotomy segmentation. European Journal of Operational Research 1999;113: 271–80.

[31] Khare V, Yao X, Deb K. Performance scaling of multi-objective evolutionary algorithms. In: Fonseca CMea, editor. EMO 2003, LNCS. Springer-Verlag; 2003. p. 376–90.

[32] Knowles J, Corne D. Approximating the nondominated front using the Pareto archived evolution strategy. Evolutionary Computation 2000;8:149–72.

[33] Knowles J, Thiele L, Zitzler E. A tutorial on the performance assessment of stochastic multiobjective optimizers. Technical report TIK-report no. 214. Computer engineering and Networks Laboratory, ETH Zurich; 2006.

[34] Laumanns M, Thiele L, Zitzler E. An efficient, adaptive parameter variation scheme for metaheuristics based on the epsilon-constraint method. European Journal of Operational Research 2006;169:932–42.

[35] Lusby R, Larsen J, Ryan D, Ehrgott M. Routing trains through railway junctions: a new set packing approach. Technical report. Informatics and Mathematical Modelling, Technical University of Denmark, DTU; 2006.

[36] Mavrotas G, Diakoulaki D. A branch and bound algorithm for mixed zero-one multiple objective linear programming. European Journal of Operational Research 1998;107:530–41.

[37] Minella G, Ruiz R, Ciavotta M. Restarted iterated Pareto greedy algorithm for multi-objective flowshop scheduling problems. Computers & Operations Research 2011;38:1521–33.

[38] Mostaghim S, Teich J. Quad-trees: a data structure for storing Pareto sets in multiobjective evolutionary algorithms with elitism. In: Jain L, Wu X, Abraham A, Jain L, Goldberg R, editors. Evolutionary multiobjective optimization. Berlin, Heidelberg: Springer; 2005. p. 81–104 [Advanced information and knowledge processing].

[39] Paquete L, Chiarandini M, Stützle T. Pareto local optimum sets in the biobjective traveling salesman problem: an experimental study. Lecture notes in economics and mathematical systems: metaheuristics for multi-objective optimization 2004;vol. 535:177–200.

[40] Paquete L, Schiavinotto T, Stützle T. On local optima in multiobjective combinatorial optimization problems. Annals of Operations Research 2007;156: 83–97.

[41] Parragh SN, Doerner KF, Gandibleux X, Hartl RF. A heuristic two-phase solution method for the multi-objective dial-a-ride problem. Networks 2009;54:227–42.

[42] Pisinger D, Ropke S. A general heuristic for vehicle routing problems. Computers & Operations Research 2007;34:2403–35.

[43] Pisinger D, Ropke S. Large neighborhood search. In: Gendreau M, Potvin JY, editors. Handbook of metaheuristics, vol. 146. US: Springer; 2010. p. 399–419.

[44] Ropke S, Pisinger D. An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows. Transportation Science 2006;40:455–72.

[45] Schaffer J. Multiple objective optimization with vector evaluated genetic algorithms. In: Grefenstette J, editor. Proceedings of the third international conference on genetic algorithms. Hillsdale: Lawrence Erlbaum; 1985. p. 93–100.

[46] Schilde M, Doerner KF, Hartl RF, Kiechle G. Metaheuristics for the bi-objective orienteering problem. Swarm Intelligence 2009;3:179–201.

[47] Schrimpf G, Schneider J, Stamm-Wilbrandt H, Dueck G. Record breaking optimization results using the ruin and recreate principle. Journal of Computational Physics 2000;159:139–71.

[48] Shaw P. Using constraint programming and local search methods to solve vehicle routing problems. In: Principles and practice of constraint programming CP98. Springer-Verlag; 1998. p. 417–31.

[49] Srinivas K. Multiobjective optimization using non-dominated sorting in genetic algorithms. Evolutionary Computation 1994;2:221–48.

[50] Tricoire F, Romauch M, Doerner KF, Hartl RF. Heuristics for the multi-period orienteering problem with multiple time windows. Computers & Operations Research 2010;37:351–67.

[51] Tsiligirides T. Heuristic methods applied to orienteering. The Journal of the Operational Research Society 1984;35:797–809.

[52] Ulungu E, Teghem J, Fortemps P, Tuyttens D. MOSA method: a tool for solving multiobjective combinatorial optimization problems. Journal of Multi-Criteria Decision Analysis 1999;8:221–36.

[53] Ulungu EL, Teghem J, Fortemps P. Heuristics for multiobjective combinatorial optimization by simulated annealing. In: Gu J, et al., editors. Proceedings of the sixth national conference on multiple criteria decision making. Windsor; 1995. p. 228–38.

[54] Van Veldhuizen D, Lamont G. Multiobjective evolutionary algorithms: analyzing the state-of-the-art. Evolutionary Computation 2000;8:125–47.

[55] Varadharajan T, Rajendran C. A multi-objective simulated-annealing algorithm for scheduling in flowshops to minimize the makespan and total flowtime of jobs. European Journal of Operational Research 2005;167:772–95.

[56] Zhang Q, Li H. MOEA/D: a multiobjective evolutionary algorithm based on decomposition. IEEE Transactions on Evolutionary Computation 2007;11:712–31.

[57] Zitzler E, Laumanns M, Thiele L. Improving the strength Pareto evolutionary algorithm for multiobjective optimization. In: Evolutionary methods for design, optimisation, and control, CIMNE, Barcelona, Spain; 2002. p. 95–100.

[58] Zitzler E, Thiele L. Multiobjective evolutionary algorithms: a comparative case study and the strength Pareto approach. IEEE Transactions on Evolutionary Computation 1999;3:257–71.

[59] Zitzler E, Thiele L, Laumanns M. Performance assessment of multiobjective optimizers: an analysis and review. IEEE Transactions on Evolutionary Computation 2002;7:117–32.