Analysis and Design of Stochastic Local Search Algorithms for the Multiobjective Traveling Salesman Problem

Luis Paquete and Thomas Stützle

IRIDIA – Technical Report Series

Technical Report No. TR/IRIDIA/2008-010

March 2008

IRIDIA – Technical Report Series
ISSN 1781-3794

Published by:

IRIDIA, *Institut de Recherches Interdisciplinaires et de Développements en Intelligence Artificielle*
Université Libre de Bruxelles
Av F. D. Roosevelt 50, CP 194/6
1050 Bruxelles, Belgium

Technical report number TR/IRIDIA/2008-010

Analysis and Design of Stochastic Local Search Algorithms for the Multiobjective Traveling Salesman Problem

Luís Paquete[1] and Thomas Stützle[2]

[1]CISUC, Department of Informatics Engineering
University of Coimbra
Coimbra, Portugal

[2]IRIDIA, CoDE
Université Libre de Bruxelles (ULB)
Bruxelles, Belgium

Abstract

Stochastic local search (SLS) algorithms are typically composed of a number of different components, each of which should contribute significantly to the final algorithm's performance. If the goal is to design and engineer effective SLS algorithms, the algorithm developer requires some insight into the importance and the behavior of possible algorithmic components. In this paper, we analyze algorithmic components of stochastic local search algorithms for the multiobjective travelling salesman problem. The analysis is done using a careful experimental design for a generic class of SLS algorithms for multiobjective combinatorial optimization. Based on the insights gained, we engineer SLS algorithms for this problem. Experimental results show that these SLS algorithms, despite their conceptual simplicity, outperform a well-known memetic algorithm for a range of benchmark instances with two and three objectives.

1 Introduction

Stochastic local search (SLS) algorithms are among the most successful techniques for tackling computationally hard problems [19]. In recent years, they have become very popular also for tackling multiobjective combinatorial optimization problems (MCOPs) [8, 36]. The currently best performing SLS algorithms for MCOPs typically involve a number of different algorithmic components that are combined into a more complex algorithm. An algorithm developer should therefore have some form of insights into the importance of these algorithmic components and know how they interact with different types of possible problem characteristics with respect to performance. Ideally, such insights are first gained to make the SLS algorithm design more informed and directed.

In this paper, we present an in-depth experimental analysis of SLS algorithms for the multiobjective traveling salesman problem (MTSP), a paradigmatic NP-hard MCOP. Our analysis is based on a sound experimental design that investigates some usual algorithmic components that can be found in a general algorithmic framework for tackling MCOPs, the scalarized acceptance criterion (SAC) search model [36]. The SAC search model mimics local search approaches that are based on the scalarization of the multiple objective functions into a single one; many such scalarizations are then tackled using local search (or exact algorithms, if the scalarized problems are efficiently solvable by such algorithms) and the resulting approximate solutions are possibly further treated. Essential components of algorithms following the SAC search model are the number of scalarizations used, the search strategy followed (for example, whether information between various scalarizations is exchanged or not), the computation time invested for tackling each of the resulting single objective problems, and various others. In fact, such a decomposition of an algorithm into its components allows to employ an experimental design perspective for the analysis of the performance of SLS algorithms: algorithmic components are seen as *factors*, that is, as abstract characteristics of an SLS algorithm that can affect the response variables such as solution quality. Designing the experiments in a careful way and analyzing them by methods from experimental design allows then to arrive at statistically sound conclusions on the importance of these components and their mutual interdependencies.

While there exist few researches where experimental designs have been used to analyze SLS algorithms for optimization problems with a single objective [40, 47], the usage of experimental designs for analyzing SLS algorithms for MCOPs is rather recent [34, 38]. One reason for this is certainly that the analysis of the outcomes of algorithms for MCOPs are difficult to compare. In fact, fundamental criticisms have been raised against the usage of many unary and binary performance measures for multiobjective optimizers [48], which also makes it difficult, if not virtually impossible, to apply the classical ANOVA-type analysis for comparing approximations to the efficient set. Instead, we employ a sound methodology that follows three steps. In a first step, the outcomes of algorithms are compared pairwise with respect to outperformance relations [17]; if these comparisons do not yield clear conclusions, we compute in a next step the attainment functions to detect *significant* differences between sets of outcomes [11, 42]. If such differences are detected, the usage of graphical illustrations is used in a third step to examine the areas in the objective space where the results of two algorithms differ [30].

Our experimental analysis allows a clear identification of the key-success components from the experimental design. For instance, the results obtained indicate the two-phase search strategy and the component-wise step proposed in [35] as two component levels that yield a significant improvement with respect to solution quality. In addition, the experimental analysis gives insights into the behavior of specific components such as the effectiveness of increasing either the number of scalarizations or the search length.

There is yet another aspect that makes the analysis through the lens of experimental design useful: The insights gained by such an in-depth analysis can be exploited to define new high-performing algorithms or at least indicate directions into which existing algorithms should be extended. Hence, the insights gained from the experimental analysis allow to yield conclusions useful towards the design and engineering of successful SLS algorithms. In fact, based on our experimental analysis, we define SLS algorithms that are assembled from the most promising levels of components we have identified; still, these algorithms remain conceptually rather simple. An extensive experimental comparison of these SLS algorithms on the MTSP with two and three objectives to a well-known *state-of-the-art* algorithm for the MTSP shows that it is very competitive or often superior.

The article is structured as follows. In section 2, we introduce basic notions on MCOPs and the MTSP. Section 3 introduces the SAC model and explains the particular components of these SLS algorithms studied in our experiments. Next, in section 4, we give an overview of the experimental design, the methodology that was used for comparing the performance of the algorithms and we describe the experimental results obtained. Finally, in section 5, we compare the performance of our SLS algorithms to a well-known state-of-the-art algorithm. We conclude in section 6.

2 Multiobjective Optimization and the MTSP

The main goal of solving MCOPs in terms of Pareto optimality is to find (all) feasible solutions that are not worse than any other solution and strictly better in at least one objective. The objective function vector for a solution $s \in S$ to an MCOP can be defined as a mapping $f : s \mapsto \mathbb{R}^Q$, where Q is the number of objectives and S the set of all feasible solutions. The following order holds for objective function vectors in $\mathbb{R}^Q$. Let $\mathbf{u}$ and $\mathbf{v}$ be vectors in $\mathbb{R}^Q$; we define the *component-wise order* as $\mathbf{u} \leq \mathbf{v}$, *i.e.*, $\mathbf{u} \neq \mathbf{v}$ and $u_i \leq v_i,\ i = 1, \ldots, Q$. In optimization, we say (i) $f(s)$ *dominates* $f(s')$ if $f(s) \leq f(s')$; (ii) $f(s)$ and $f(s')$ are *non-dominated* if $f(s) \not\leq f(s')$ and $f(s') \not\leq f(s)$. We use the same notation and wording among solutions if these relations hold between their objective function vectors.

A solution $s \in S$ is said to be a *Pareto global optimum solution* if and only if there is no $s' \in S$ such that $f(s') \leq f(s)$. There may be more than one Pareto global optimum solution; a *Pareto global optimum set* is the set $S' \subseteq S$ that contains *only* and *all* Pareto global optimum solutions. We call the image of the Pareto global optimum set in the objective space the *efficient set*. In most cases, solving an MCOP in terms of Pareto optimality would correspond to finding solutions that are representative of the efficient set.

The optimization problem handled in this study is the multiobjective traveling salesman problem (MTSP), which is defined as follows: Given Q, a set C of n cities, and distance vectors $d(c_i, c_j) \in \mathbb{N}^Q$ for each pair of cities $c_i, c_j \in C$, the goal is to find a tour in C, that is, a permutation $\pi : [1..n] \rightarrow [1..n]$, such that the length of the tour, that is,

$$f(\pi) = d\left(\{c_{\pi(n)}, c_{\pi(1)}\}\right) + \sum_{i=1}^{n-1} d\left(\{c_{\pi(i)}, c_{\pi(i+1)}\}\right)$$

is "minimal". In this paper, "minimal" is understood in terms of Pareto optimality. The MTSP is known to be NP-hard [41]; additionally, it is known that the lower bound on the expected size of the efficient set for the MTSP is an exponential function of the instance size [9].

The MTSP was chosen for three main reasons. Firstly, its single objective counterpart, the traveling salesman problem (TSP), is one of the best studied NP-hard combinatorial optimization problems and it has been intensively used as a test-bed for experimenting new algorithmic ideas [25], including many SLS algorithms. Hence, experimental results obtained for the multiobjective version may also be interpreted in light of the experience on the performance of these techniques for the single objective case. Secondly, despite the fact that the small instances of the single-objective TSP can be solved in a few seconds to optimality by exact algorithms such as the concorde solver (`http://www.tsp.gatech.edu/concorde`), there are two facts that limit their use under fixed time constraints: the typically large variability in the computation times and the potentially very large number of solutions in the efficient set [9]. Thirdly, significant research efforts have been targeted towards applying SLS algorithms to this problem and it has been studied from several different perspectives: from an approximation [1, 7], local search [2, 16, 23], theoretical [9], and experimental [4] point of view; some related problems have been studied in [27, 43].

3 The Search Model and Algorithmic Components

A large number of SLS algorithms have been proposed for MCOPs. Many of these algorithms can be classified as following one of two main search models, the scalarized acceptance criterion (SAC) and the component-wise acceptance criterion search (CWAC) search model, or some hybrid thereof [36]. In this article, we analyse the influence of generic components that together mimic the underlying search principles of the SAC search model. (An analysis of generic components of the CWAC search model can be found in [33]; the algorithms following the CWAC model were found to be inferior to those of the SAC model and, hence, here we present only the analysis concerning the more successful model.) Essentially, the SAC search model comprises approaches that use the value returned for solving a scalarization of the objectives for deciding upon the quality of solutions. For a reasonable approximation to the efficient set, it is well-known that it is necessary to solve a number of such scalarizations.

As one representative of the SAC search model, we examine the straightforward approach that uses several scalarizations of the objective function vector and tackles these with an underlying algorithm for the resulting single objective problem, an approach which underlies many earlier proposed algorithms [6, 13, 17, 23, 46]. We follow the well-known principle of defining scalarizations of the objective function vector with respect to a weighted sum. Hence, the scalarized (single) objective function is defined as

$$f_\lambda(s) = \sum_{q=1}^{Q} \lambda_q f_q(s), \tag{1}$$

where $\lambda = (\lambda_1, \ldots, \lambda_Q)$ is a weight vector. Typically, λ is normalized such that its components sum to one. We follow this convention and each λ we use is an element from the set of normalized weight vectors Λ given by

$$\Lambda = \{\lambda \in R^Q : \lambda_q > 0, \sum_{q=1}^{Q} \lambda_q = 1, q = 1, \ldots, Q\}. \tag{2}$$

Algorithms following the SAC model then solve a number of scalarized problems that are obtained by different weight vectors. The resulting scalarized problems could be solved by any algorithm for the resulting single-objective version; in our case, we apply an effective SLS algorithm for the TSP, which is described later in more detail. In the following subsections, we describe several algorithmic components for the SAC model that are analyzed in this article. For each component at least two and at most three levels are studied in the experimental design; although for several components more options would be possible and interesting, the number of levels was kept restricted to limit the exponential increase of the number of experiments.

3.1 Component: Search strategy

The search strategy determines the series of scalarized problems that are defined and tackled. In particular, this concerns the strategy for defining the sequence of weight vectors and how information is transferred from one scalarized problem to another one. We consider two different search strategies.

Restart strategy. The probably most straightforward strategy is to use different weight vectors and to not transfer the results from one scalarized problem to another one. Such as strategy is obtained, for example, by starting the search process for each scalarization from a random initial solution (or, alternatively, by some known construction heuristic that does not use information from previous runs). We call this approach the `Restart` strategy and its pseudo-code is given in Algorithm 1; it results in multiple, independent runs of the single objective SLS algorithm. The procedure SLS at the third line is the underlying SLS algorithm that tackles the problems obtained by weight vector λ_i. Since A could have dominated solutions at step 5, only the non-weakly dominated solutions are kept; this is implemented by the sub-procedure Filter.

Algorithm 1 `Restart` search strategy

```
for all weight vectors $\lambda \in \Lambda$ do
    $s$ is a randomly generated solution
    $s' = \mathsf{SLS}(s, \lambda)$
    Add $s'$ to Archive
Filter Archive
return Archive
```

A set of m weight vectors is used by the `Restart` strategy. Here, we assume that each component of the weight vector has a value $i/z, i = 0, \dots, z$, where z is a parameter, and the sum of the components is equal to one, as required by Equation 2. Since this set of weight vectors can be seen as the set of all compositions of z in Q parts, we have that $m = \binom{z+Q-1}{Q-1}$.

2phase strategy. A different possibility is to transfer results from one scalarization to another one. Here, we adopt the `2phase` strategy, which was proposed in [35]. In a first phase, a high quality solution for one objective is generated. This solution is the starting solution for the second phase that solves a sequence of scalarizations of the objective function vector. In this sequence, the initial solution for a scalarization i is the one that is returned from the previous scalarization $i - 1$; the first scalarization of the second phase is initialized with the solution returned from the first phase. A pseudo-code of the `2phase` search strategy is shown in Algorithm 2. Note that the first and the second phase may make use of two distinct SLS algorithms, which is indicated in Algorithm 2 by SLS_1 and SLS_2.

Concerning the definition of the sequence of weight vectors that define each scalarization, several strategies may be followed. Here, we adopt a *minimal change* strategy between two successive weight vectors to define this sequence. It can be built for two objectives by generating $m = z$ weight vectors such that $\lambda_i = (1 - i/z, i/z), i = 1, \dots, m$, if the first objective is the one that is optimized in the first phase. For more than two objectives, we define this sequence such that two successive weight vectors differ only by $\pm 1/z$ in any two components. Thus, a minimal change is incurred between components of two successive weight vectors generalizing the biobjective case. To generate such a sequence, an algorithm for generating compositions of z into Q parts can be used. In our particular case, we need a *combinatorial Gray code* for this task, which can be generated by the *Gray code for compositions* [28].

Finally, one may run the `2phase` strategy considering different orders of the objectives. One possibility would be to run it once for each permutation of the Q objectives, that is, $Q!$ times. Computationally less expensive is to consider only the combinations of each pair of objectives, totalizing $\binom{Q}{2}$ runs, or to apply just one run for each objective.

Algorithm 2 `2phase` search strategy

```
$s$ is a randomly generated solution
$s' = \mathsf{SLS}_1(s)$                    /* First phase */
for all weight vectors $\lambda \in \Lambda$ do
    $s = s'$
    $s' = \mathsf{SLS}_2(s, \lambda)$               /* Second phase */
    Add $s'$ to Archive
Filter Archive
return Archive
```

3.2 Component: Number of Scalarizations

The number of scalarizations is defined by the parameter z. It is expected that an increase of the number of scalarizations would increase also the number of different (non-dominated) solutions returned. However, how much the number of solutions grows when increasing the number of scalarizations is not clear in advance. Therefore, we consider the number of scalarizations as a numerical parameter, that is, also as an algorithmic component, whose influence is studied in the experimental part.

3.3 Component: Neighborhood structure

In our analysis, we study two main components of the underlying SLS algorithm. Both are related to the quality of the solutions it returns. The first component is the neighborhood structure that is used; it identifies which solutions are neighbored and it has a significant influence on the performance of local search algorithms. The typical trade-off is that the larger the neighborhood, the better quality are the solutions that are found by, for example, iterative improvement algorithms; however, with the neighborhood size also increases the computation time that is required to find improving neighbors and, in the case of iterative improvement, a local optimum.

3.4 Component: Search length

The second component of the underlying SLS algorithm we study is the number of iterations this algorithm is run. The reason for studying this component is that by increasing the number of iterations, the final solution quality returned by the SLS algorithm tends to increase, but at the same time does also the computation time. In other words, there is a trade-off between these two criteria and a good compromise needs to be found for the design of an algorithm following the SAC model.

3.5 Component: Component-wise Step

The number of solutions returned is bounded by the number of compositions of z in Q parts. One possibility for increasing this number is by accepting, for each scalarization, non-dominated solutions in the neighborhood of the solution returned by the underlying single-objective SLS algorithm. We call this additional component *component-wise step*, which was also used in [35]. In our particular case, this step uses the neighborhood that is defined by the neighborhood component.

y

4 Experimental Analysis

4.1 Experimental design

The experimental design studied all the five algorithm factors that were described in the previous section plus two factors concerning the MTSP instances, namely the instance type and the instance size.

4.1.1 MTSP instances

For the experimental part of the study, we have generated MTSP instances of different size and using different ways to generate the distances between the nodes. We used the random instance generator available from the 8th DIMACS Implementation Challenge site[1] and for each objective, one instance matrix was generated. We consider only two objectives for the experimental analysis of the algorithm components, while the comparison with a *state-of-the-art* SLS algorithm for the MTSP includes also instances with three objectives. Three types of biobjective instances were generated.[2]

- *Random Uniform Euclidean* (RUE) instances, where each component of the distance vector assigned to an edge is generated as usual in RUE instances: each distance value corresponds to the Euclidean distance between two points in a two-dimensional plane rounded to the next integer; the coordinates of each point are integers that are uniformly and independently generated in the range $[0, 3163]$.
- *Random distance matrix* (RDM) instances, where each component of the distance vector assigned to an edge is chosen as an integer value taken from a uniform distribution in the range $[0, 4473]$.
- *Mixed* instances, where one objective assigns distances to the edges as in RUE instances while the other assigns distances as in RDM instances.

The range of edge lengths for the RUE instances was chosen in order to meet the range of values of the Krolak/Felts/Nelson instances available in TSPLIB (files with prefix `kro`). The range of the edge lengths for the RDM instances and the RDM objective of the mixed instances were chosen in order to have a range similar to the one of the RUE instances (note that $\lfloor\sqrt{2 \cdot 3163^2} + 0.5\rfloor = 4473$).

[1]The generator is available at `http://www.research.att.com/~dsj/chtsp/download`.

[2]The instances are available at `http://eden.dei.uc.pt/~paquete/tsp/`.

Components	Levels
Search Strategy	{`Restart`, `2phase`}
Number of scalarizations	{n, $5n$, $10n$}
Neighborhood structure	{`2-exchange`, `3-exchange`}
Search length	{0, 50, 100 }
Component-wise step	{`True`, `False` }

Table 1: List of algorithmic components and the corresponding levels that we considered for our experimental setup.

The instance sizes considered were $n \in \{100, 300, 500\}$. Thus, the instances we tackle are larger than most that are considered in the literature, where mainly experimental results are available for instances with less than 300 cities. For each instance size and type of instance, three instances were generated, resulting in a total of 27 instances.

4.1.2 Algorithmic component levels

For each of the factors concerning algorithmic components, two or three levels have been studied. A summary of the components and their associated levels studied is given in Table 1. Necessary details on the components are explained next.

Search strategy. If the search strategy `2phase` is chosen, the first phase will optimize the first objective and the solution returned is also the starting solution for the second phase. For RUE and RDM instances, the first phase of `2phase` consisted in running an iterated local search (ILS) algorithm, which is described below in some more detail, for 50 iterations only for the first objective; this resulted in a high quality solution for the single objective case. For the mixed instances, we considered two variants of the `2phase` strategy: $\texttt{2phase}_E$ starts the first phase optimizing only the objective defined by the Euclidean distance matrix; $\texttt{2phase}_R$ starts optimizing the objective defined by the RDM distance matrix. Using these two versions of the `2phase` strategy, we can thus also analyze the dependence of the final performance on the structure of the objective that is used for generating the initial solution for the second phase. Since in the mixed instances the solutions found in the efficient set had a lower range of objective function values for the RUE objective than for the RDM objective (despite the fact that we tried to equalize the range of the edge weights), we equalized the ranges of both objectives by multiplying the ith component of the objective function value vector by a range equalization factor F_i [44] that for each objective i is

$$F_i = \frac{R_i}{\sum_{j=1}^{Q} R_j},$$

where Q is the number of objectives and R_i is the range of objective function values for the objective i. We computed an approximation to the range of the efficient set as follows. First, we run an iterated local search (ILS) algorithm [45] 10 times for each objective; then, given the best solutions to the first and the second objective, s_1 and s_2, respectively, the ranges are computed as $R_1 = f_1(s_2) - f_1(s_1)$ for the first objective and as $R_2 = f_2(s_1) - f_2(s_2)$ for the second objective.

Number of scalarizations. For the number of scalarizations, three levels have been studied, n, $5n$, and $10n$, where n is the number of cities in the MTSP instance.

Neighborhood structure. We consider two standard TSP neighborhood structures, the `2-` and `3-exchange` neighborhoods. (In general, two solutions are neighbored in the k-exchange neighborhood if they differ by at most k edges.) The iterative improvement algorithms we use for each scalarization make use of the usual TSP speed-up techniques and measure whether a neighboring solution improves over a current one using the weighted sum of the objectives. Clearly, one could also use more complex neighborhood structures like the ones used in the Lin-Kernighan local search algorithms [29]; however, we restricted to the `2-` and `3-exchange` neighborhoods for the sake of limiting the exponential increase of the number of configurations with the number of levels.

Search length. On top of the resulting two iterative improvement algorithms, we used an iterated local search (ILS) method [31]. The usage of a general-purpose SLS method allows to set larger search lengths and in this way to intensify the search for each scalarization. (In fact, the ILS method forms the basis for most high-performing SLS algorithms for the TSP.) An algorithmic outline of ILS is given in Algorithm 3. The perturbation used in the ILS algorithm is a random double-bridge move and the acceptance criterion accepts a new solution only if it improves over the previous one. For the ILS algorithm, we have used the code provided at `http://www.sls-book.net/`. Since the ILS algorithm uses the iterative improvement algorithm as its subsidiary local search procedure, the application of the iterative improvement algorithm corresponds to "zero iterations" of the ILS algorithm. Hence, the three different levels of the *search length* component can be indicated as 0, 50, and 100 iterations of ILS, respectively, which we denote in the following as ILS(0), ILS(50), and ILS(100).

Component-wise step. The effect of the component-wise step was explicitely analyzed only for instance size 100; in fact, on all instance sizes the component-wise step has a (strongly) positive effect and, hence, we simply used it on the large instances always. This has the side-advantage of reducing the number of experiments on the instances of size 300 and 500 by a factor of two and, thus, saving significant computational effort.

Algorithm 3 *Iterated Local Search.* Algorithmic outline of an iterated local search algorithm as used in the experimental setting. `no_iterations` is a parameter that defines the number of times the main loop of the algorithm is executed. If `no_iterations` $= 0$, the algorithm corresponds to a single application of iterative improvement.

```
input: $s_0$, an initial solution
input: no_iterations; number of iterations of main loop
$s^* :=$ IterativeImprovement($s_0$)
for $i := 1$ to no_iterations do do
    $s' :=$ Perturbation($s^*$)
    $s^{*\prime} :=$ IterativeImprovement($s'$)
    $s^* :=$ AcceptanceCriterion($s^*, s^{*\prime}$)
return $s^*$
```

4.2 Performance Assessment Methodology

Assessing the performance of algorithms for MCOPs is by far more complex than in the single-objective case and a number of serious problems, in particular of unary performance indicators, have been described [48]. Our experimental analysis is based on a three step evaluation that avoids these known drawbacks. In a first step we use the *better* relations, which provide the most basic assertion of performance; the second step computes attainment functions and tests the equality of the attainment functions [15]; the third step consists in detecting the largest differences of performance in the objective space between pairs of algorithms. Most aspects of this three-step experimental analysis were described in [34, 30] and are here summarized for the sake of comprehensibility of the remainder of the paper.

Step 1: Better Relations

A set of points A is *better* than a set of points B if every point of B is dominated or equal to any point of A. This relation was introduced in [17] as one of the outperformance relations that can be established between pairs of outcomes of SLS algorithms for MCOPs. Thus, as a first step, we count how many times each outcome associated to each level of a component is *better* than the ones from another level of the same component. However, we restrict the comparison of outcomes to those that were produced *within* the same levels of other components in order to reduce variability. This allows us to detect if some level is *clearly* responsible for a good or bad performance. If no clear answers are obtained from this first step, we can conclude that the outcomes are mostly *incomparable*, that is, neither A is *better* than B nor vice versa. Since then we do not know to what extent they really differ, we test the equality of their attainment functions.

Step 2: Attainment Functions

In Fonseca and Fleming [11], the performance of an SLS algorithm for multiobjective problems is associated to the probability of attaining (dominating or being equal to) an *arbitrary point* in the objective space in one single run. This function is called *attainment function* in [15] and it can be seen as a generalization of the distribution function of solution cost [19] to the multiobjective case. These probabilities can be estimated empirically from the outcomes obtained in several runs of an SLS algorithm by the *empirical attainment function* (EAF). Then, we can formulate statistical hypotheses and test them based on the EAFs of several algorithms for a certain problem instance. A suitable test statistic for the comparison of two algorithms is the maximum absolute distance between their corresponding EAFs, analogous to the Kolmogorov–Smirnov statistic [5]. For the case of $k > 2$ algorithms, we choose the maximum absolute distance between the k EAFs, analogous to the Birnbaum–Hall test [5]; if the global null hypothesis of equality is rejected, we test the equality between each pair of EAFs, where the p-values are corrected by Holm's procedure [20]. Since the distribution of these test statistics is not known, permutation tests [14] based on the above test statistics have to be performed [42]. The permutation procedure has to be changed according to the experimental design chosen. For instance, in the presence of several factors, restricted randomizations [14], as done in [34] in a similar context, can be applied; for testing the main effects of each component, we allow permutations of the outcomes *between* different levels of the component of interest, but *within* the same levels of the other components.

Step 3: Location of Differences

If the previous analysis indicates that the null hypothesis of equality of the attainment functions should be rejected, the largest performance differences can be visualized by plotting the points in the objective space with a large absolute difference of the EAFs. In fact, large has a subjective meaning; here, we plot the points whose absolute differences were above or equal to 20%, assuming that lower values are negligible.[3] Since the sign of the difference at each point gives information about which algorithm performed better at that point, we may plot positive and negative differences separately, if differences in both directions exist.

Figure 2 on page 15 illustrates the main idea. Each of the two plots give the differences of the EAFs associated to two algorithms that were run several times on one instance. The lower line on each plot is a lower bound on the efficient set,[4] while the upper line connects the set of points attained by all runs of both

[3]The minimum number of outcomes that were used for statistical tests in this thesis were 10 (5 runs associated to each level of a factor); thus, a difference of 20% corresponds also to the minimum difference that can be observed in these comparisons.

[4]The lower bound is used simply as a visual reference when plotting the differences with respect to the empirical attainment functions. We use a lower bound based on the solution of the 2-matching problem; it yields a lower bound approximately at 14% of the optimum for the TSP [39]. (Note that better quality lower bounds exist, but this is for our purposes

algorithms. On both plots are shown the regions where the EAF of Algorithm 1 (using `2phase` strategy) is larger by at least 20% than that of Algorithm 2 (using `Restart` strategy); the observed differences are encoded using a grey scale–the darker the stronger are the differences. In this case, no point in the EAF of Algorithm 2 was larger than the corresponding one of Algorithm 1. If differences in favor of each of the algorithms occur, the positive differences in favor of each algorithm can be plotted side-by-side, as it is done in Figure 1 on page 14.

4.3 Experimental Results

Each configuration resulting from any of the possible combinations of levels of the factors, as described in Section 4.1, was run five times on an AMD Athlon(TM) 1.2 GHz CPU, 512 MB of RAM under Suse Linux 7.3. We have permuted randomly the original order of the runs to remove a possible bias. In the following, we discuss the results of the analysis for each SLS component under study. Each permutation test for testing hypotheses on the equality of EAFs used 10 000 permutations and the significance level was set to $\alpha = 0.05$. Due to space restrictions, we only show the most relevant plots of the locations of the differences as explained in Section 4.2; a full collection of the results comprising all the data on the comparisons and all plots is available at `http://eden.dei.uc.pt/~paquete/mtsp`.[5]

4.3.1 Component: Search Strategy

The results in Table 2 with respect to the better relations indicate that the `2phase` strategy performs slightly better than the `Restart` strategy for larger instances and that the difference is more relevant on RDM instances. In addition, the null hypothesis of equality of the EAFs was always rejected. Hence, the search strategies behave statistically different with respect to the corresponding EAFs in the instances tested.

Figures 1 and 2 indicate the location of differences above 20% between the search strategies. The two plots of Figure 1 indicate that the `Restart` strategy covers a wider part of the trade-off than the `2phase` strategy on the RUE instances of size 100, though with only a small difference (which is reflected by the fact that the differences are almost imperceptible); the latter performs better only towards the first objective, where the first phase terminated. However, as instance size increases in RUE instances and RDM, the `2phase` strategy performs clearly better than the `Restart` strategy, as shown in the plots of Figure 2.

not important.) For our particular case, we solved the 2-matching problem using as input a matrix resulting from the weighted sum of distances assigned to each edge of an MTSP instance. This procedure is repeated for 5 000 maximally dispersed weight vectors and the lower bounds have been computed using CPLEX.

[5]Note that the computation of all the experimental results including the execution of the hypothesis tests took more than half a CPU year; in fact, the exponential increase of the number of experiments was one reason for limiting the experiments to a small number of levels for each factor.

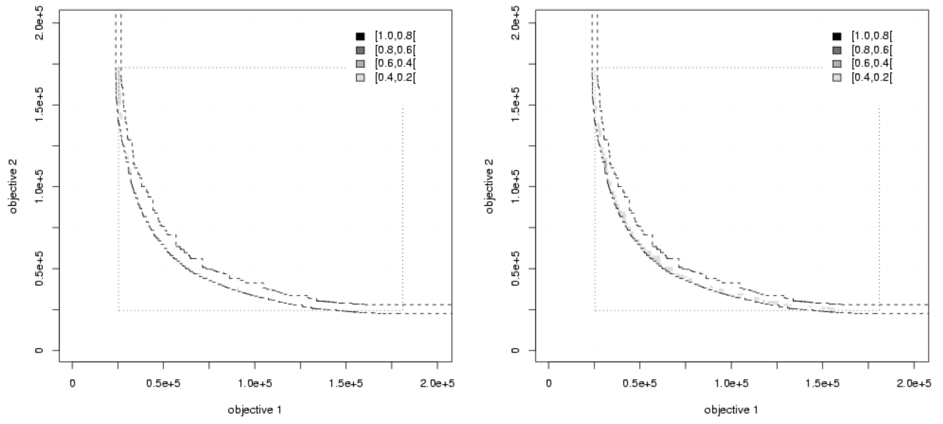

Figure 1: Location of differences between the `2phase` and the `Restart` search strategy in favor of the former (left) and in favor of latter (right), for RUE instances of size 100. Note that the differences between the two strategies in this case are all minor, that is, in the range of $[0.2, 0.4[$, and are therefore almost imperceptible.

Finally, the comparisons on the mixed instances have shown interesting tendences: When comparing $\texttt{2phase}_E$ and $\texttt{2phase}_R$, each has advantages towards the objective that is optimized in the first phase and the observed differences between the two are roughly the same across the various instance sizes. Differences in favor of the `Restart` search strategy for instance size 100 are located in the center of the trade-off; however, on larger instances, no differences above 20% in favor of the `Restart` strategy were found.

4.3.2 Component: Component-wise Step

The comparison based on the better relation with respect to the use or not of the component-wise step always resulted in incomparable cases, but the null hypothesis with respect to the equality of EAFs was always rejected. Hence, there are always significant differences between using or not the component-wise step. The location of differences above 20% clearly indicates that the use of this step yields a significant advantage, as shown in the left plot of Figure 3. However, this advantage is not constant over all types of instances tested: the differences are stronger for RUE instances than for RDM instances. In addition, the differences for mixed instances lie more towards the Euclidean objective.

Concerning the computation time, it is remarkable that the addition of the component-wise step increases the computation time by only about 1%, which is negligible in most applications. Furthermore, the number of non-dominated solutions is increased by a factor of about six for RUE instances and by a factor of about three for RDM instances.

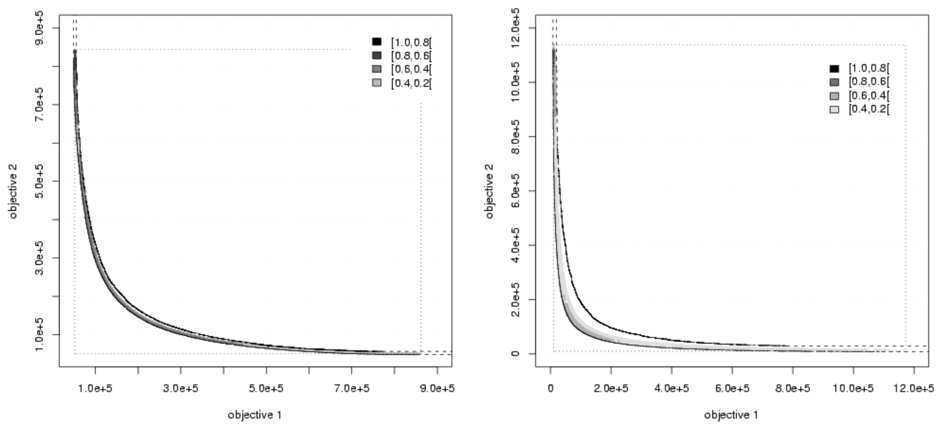

Figure 2: Location of differences between search strategies for an RUE (left) and an RDM instance (right) of size 500. All differences are in favor of the `2phase` strategy.

	RUE	RDM	Mixed	
size	`2phase`	`2phase`	`2phase`$_E$	`2phase`$_R$
100	0.0%	6.4%	0.0%	0.0%
300	4.1%	31.6%	5.2%	2.1%
500	10.1%	31.7%	15.8%	6.9%

Table 2: It gives the percentage of the pairwise comparisons in which the `2phase` search strategy (for mixed instances, `2phase`$_E$ and `2phase`$_R$, respectively) was better than the `Restart` strategy. In no comparison, `Restart` was found to be better than `2phase` (or `2phase`$_E$ and `2phase`$_R$, respectively).

4.3.3 Component: Neighborhood Structure

The results based on the better relation indicated that using a `3-exchange` neighborhood results in significantly better performance than `2-exchange` with the advantage of `3-exchange` over `2-exchange` increasing strongly with instance size; the advantage of `3-exchange` is most evident for the RDM instances. Table 3 gives a summary of the observed percentages of `3-exchange` being better than `2-exchange` for the different instance sizes and the different instance types. Given these strong differences, clearly also the null hypothesis with respect to the equality of the EAFs was always rejected. The differences above 20% were always in favour of `3-exchange` and the differences were more pronounced for larger instances (see right plot of Figure 3). Hence, this result is analogous to the relative behavior between these neighborhoods in iterative improvement algorithms for the single-objective TSP [39, 25, 19].

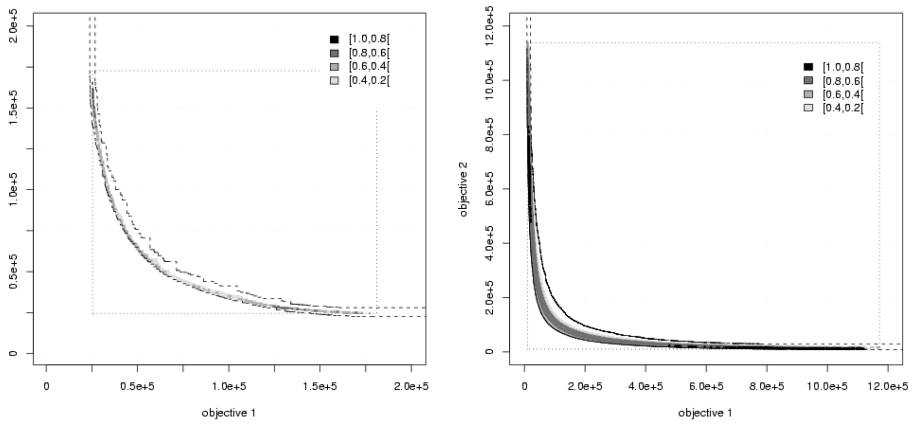

Figure 3: Location of differences between using or not the component-wise step in favor of the latter for an RUE instance of size 100 (left) and between the `2-exchange` and the `3-exchange` neighborhood in favor of the latter for an RDM instance of size 500 (right).

Size	RUE	RDM	Mixed
100	1.2%	50.3%	6.5%
300	43.0%	67.4%	22.3%
500	53.1%	75.0%	26.2%

Table 3: It gives the percentage of the pairwise comparisons in which `3-exchange` is better than `2-exchange`. In no comparison `2-exchange` was found to better than `3-exchange`.

4.3.4 Component: Search length

As said before, the search length defines the number of iterations for a single execution of the ILS algorithm, denoted by ILS(i). According to the use of the better relation, we observed that ILS(50) and ILS(100) perform better than ILS(0), that is, iterative improvement, and that the frequency of better performance is higher for instances of size 300 and 500 than for those of size 100. The relative performance seems similar across all types of instances, though less emphasized for mixed instances. See Table 4 for more details. The results also indicate that the comparisons between the outcomes obtained by ILS(50) and ILS(100) are incomparable.

The statistical tests on the equality of the EAFs all clearly indicate the rejection of the null hypothesis for all instances. Thus, any increase of the number of iterations from 50 to 100 results still in statistically significantly better performance. However, the examination of the location of the differences above 20%

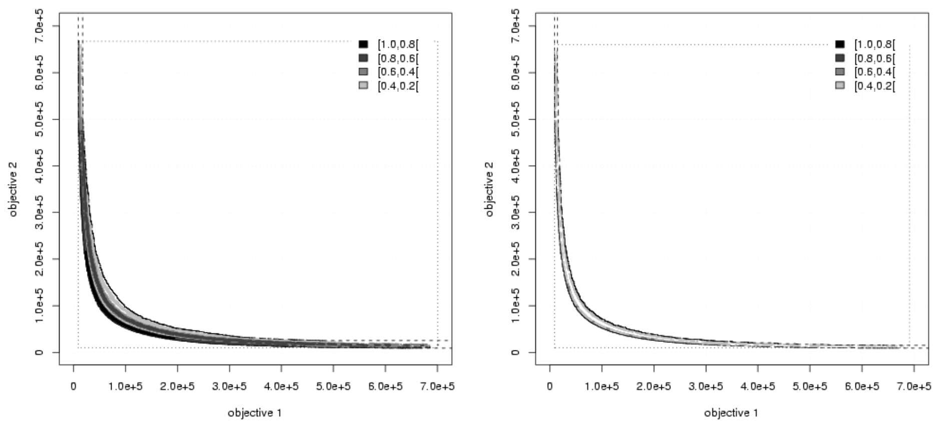

Figure 4: Location of differences between iterative improvement and ILS(50) in favor of the latter (left) and between ILS(50) and ILS(100) iterations in favor of the latter (right) for an RDM instance of size 300.

	RUE		RDM		Mixed	
size	0 vs. 50	0 vs. 100	0 vs. 50	0 vs. 100	0 vs. 50	0 vs. 100
100	15.1%	20.4%	42.6%	54.9%	22.8%	26.6%
300	65.3%	80.0%	67.0%	77.7%	34.6%	52.8%
500	71.8%	79.9%	69.0%	74.1%	40.0%	49.7%

Table 4: It gives the percentage of the pairwise comparisons in which a search length of 0 is worse than a search length of 50 and 100. In no comparison, a search length of 0 was better than 50 or 100. The pairwise comparisons between search lengths 50 and 100 resulted always in incomparable cases.

indicates that, for all the instances tested, the major leap in performance is given by moving from ILS(0) to ILS(50), while moving from ILS(50) to ILS(100) yields somewhat less pronounced but still significant differences. For an illustration of this behavior, we refer to Figure 4.

4.3.5 Component: Number of Scalarizations

Differently from an increase of the search length, an increase of the number of scalarizations does not correspond to an evidently better performance with respect to the better relation; as can be seen in Table 5, some minor evidence for improved performance is only found for large RDM and mixed instances. However, the null hypothesis of equality with respect to the EAFs is always rejected for any instance, which means that an increase of the number of scalarizations results in a significant effect. The location of differences above 20% indicates

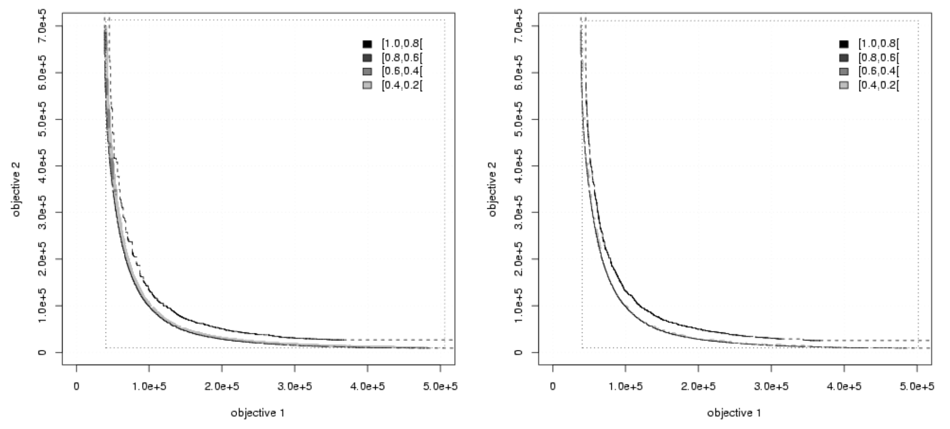

Figure 5: Location of differences between n and $5n$ scalarizations in favor of the latter (left) and between $5n$ and $10n$ scalarizations in favor of the latter (right) for a mixed instance of size 300.

	RUE		RDM		Mixed	
size	n vs. $5n$	$5n$ vs. $10n$	n vs. $5n$	$5n$ vs. $10n$	n vs. $5n$	$5n$ vs. $10n$
100	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
300	0.0%	0.0%	8.3%	11.9%	0.9%	7.3%
500	0.0%	0.0%	7.0%	10.5%	1.1%	7.6%

Table 5: It gives the percentage of the pairwise comparisons in which a number of scalarizations j is better than a number of scalarizations i (indicated by i vs. j). In no comparison, a number of scalarizations $i < j$ was better than j.

that the performance differences are not very pronounced. While the differences between n against $5n$ scalarizations are still rather clearly visible, as shown in Figure 5 on the left plot, the differences between $5n$ and $10n$ scalarizations are almost imperceptible (see right plot).

4.4 Summary

The main insights from the extensive experimental design analysis for the SAC search model applied to the MTSP are the following. A substantial gain in solution quality can be obtained by choosing an underlying high performing SLS algorithm. Two ways of improving the performance of the underlying SLS algorithm have been studied: the underlying neighborhood (solution quality is known to improve considerably already for the single objective case when moving from `2-exchange` to `3-exchange` neighborhood in an iterative improvement algorithm) and the search length, here defined by the number of iterations of the underlying ILS algorithm. In fact, these insights would suggest that a further improvement of the performance might be expected when moving to effective implementations of the Lin-Kernighan algorithms as provided by Helsgaun [18] or the concorde library [3]. (This is the case because for the single-objective TSP, the Lin-Kernighan algorithm is known to reach better quality solutions when comparing iterative improvement algorithms; in fact, this same ranking transfers to the case once the iterative improvement algorithms are included into an ILS algorithm [25, 26, 19].)

The component-wise step was also shown to have a significant impact on the final solution quality, as we observed on all instances studied. In addition, the computational overhead caused by its introduction seems to be minor at least in the biobjective case studied here: on average it leads to an increase of the computation time by approximately one percent. Hence, concerning computation time the impact of adding that component-wise step is much less than when moving from the `2-exchange` to `3-exchange` neighborhood, which actually increases the computation time significantly.

Concerning the choice of the search strategy, there is a clear interaction between the search strategy and the type of instance. For small RUE instances with 100 cities, the `Restart` strategy has slight advantages over `2phase`. However, for larger mixed and Euclidean instances, and for all RDM instances tested, the `2phase` strategy is clearly preferable, often by a large margin. In fact, the experimental analysis of the SAC search model indicated that instance features play a strong role in the performance of the algorithms under study; it is therefore expected that those features are also relevant in the performance of many other SLS algorithms as well.

The impact of the number of scalarizations seems to have the smallest impact, at least when judging from the better relations. In the plots of the differences, the step from n to $5n$ scalarizations was most noticeable, while further increasing it to $10n$ gave no further strong advantages.

5 Comparison with a State-of-the-Art Algorithm

The insights gained from an extensive study of a class of algorithms through experimental designs may, beyond the scientific insights gained, also be useful to define new high-performing algorithms. Here, we show that, indeed, this step can effectively be done by defining such algorithms and then comparing them to a multiobjective memetic algorithm called MOGLS. This algorithm was proposed by Jaszkiewicz [23], who kindly provided us the source code of this algorithm, and in earlier studies it was shown to outperform other SLS algorithms for the MTSP for two and three objectives [23]: the algorithms to which MOGLS was compared include MOGA [10], MOSA [46], and Ishibushi and Murata's Memetic Algorithm [22].

MOGLS works as follows. It initializes two archives CS and A with l solutions obtained from runs of an iterative improvement algorithm based on the `2-exchange` neighborhood with respect to a weighted sum scalarization with randomly generated weights. Then, it iterates r times over the following two steps. First, it chooses two among the best m solutions in A with respect to a weighted sum based on randomly generated weights; these two solutions are then *recombined* by the *distance-preserving crossover* [12]. Next, it applies a different iterative improvement algorithm based on the `2-exchange` neighborhood to the new recombined solution using the same weighted sum scalarization; the resulting local optimum s^* is added to archive CS if it is better than the worst solution among the m best solutions according to the weight vector considered; finally s^* is added to archive A, if no solution dominates it and the archive A is updated. The performance of this approach depends on parameters m, l and r. These two steps (recombination and local improvement) are repeated for $r \cdot l$ iterations, thus generating a total of $(r+1) \cdot l$ local optima between the initialization and the following iterations.

For our experiments, we follow the parameter settings proposed in [23] as far as possible: we set $m = 16$, the number of iterations $r = 50$, which was the maximum value tested earlier; for l, we used the value 142 for instances of size 100 with two objectives, but for instances of size 300 we extrapolate it linearly, which resulted in $l = 278$.

The configurations of our SLS algorithms were chosen according to the main insights from the experimental design. The only exception is that we use only local search based on the `2-exchange` neighborhood, since also MOGLS uses a local search in the `2-exchange` neighborhood and using the `3-exchange` neighborhood would clearly bias the results in our favor. For the comparison, we tested the two approaches on the RUE and RDM instances with 100 and 300 cities. We used the following configurations: All SLS algorithms use (i) the component-wise step, (ii) 100 iterations of the ILS algorithm, (iii) $10n$ scalarizations, and (iv) the `2-exchange` neighborhood. Regarding the search strategy, we use always the `2phase` strategy, with the only exception being the RUE instances with 100 cities: for these instances, our experimental analysis indicated slightly better performance with the `Restart` strategy and, hence, we follow our conclusions of the experimental analysis. Each of the algorithms was run 10 times on a single CPU of a computer with two AMD Athlon(TM) 1.20 GHz CPUs with 512 MB of RAM running under Suse Linux 7.3. For measuring the computational effort spent by the algorithms, we use the number of times a neighboring solution is evaluated (recombinations in case of MOGLS, and perturbations in the ILS were not counted). This was done, because the implementations were not done using the same programming languages and the very same data structures and, hence, measuring CPU time would have been unfair. (We verified that our code was actually much faster concerning CPU time than MOGLS and, hence, in this way we also avoid the bias in favor of our code that would occur if we stop both algorithms at a same CPU time.)

Given that most of the outcomes returned by the two algorithms were incomparable in some preliminary experiments, we decided to directly apply the statistical tests at the 5% significance level for checking the null hypothesis of equality between attainment functions: for all experiments, the null hypothesis was always rejected. The location of the differences above 20% showed a clear better performance of our SLS algorithms over MOGLS, except in the RUE instance with 100 cities and two objectives.[6] On this instance, the differences were rather small, except towards the improvement of the second objective where our configuration performs better (see Figure 6). The plots show that the differences are larger and more spread in favor of our SLS algorithm, though we can notice that there are still regions of the objective space where MOGLS performs slightly better. However, for all RDM instances, and for all RUE instances with 300 cities, we found only differences in favor of our approach. (For an example, see Figure 7.)

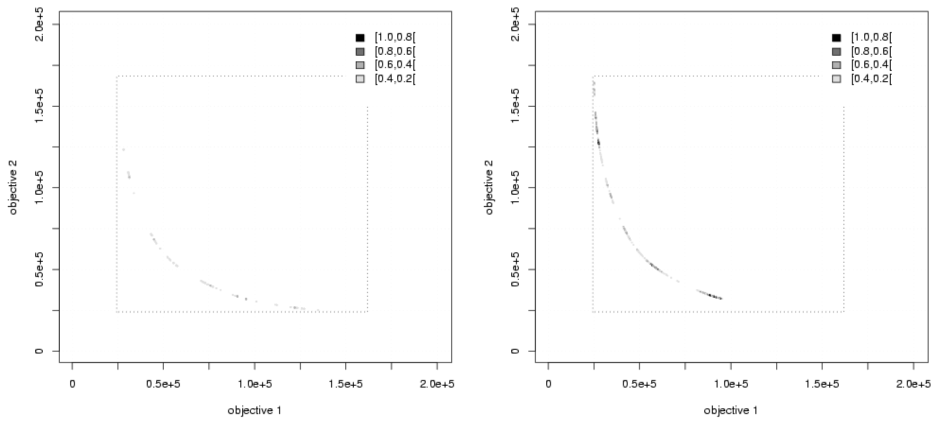

Figure 6: Location of differences between MOGLS and our SLS algorithms in favor of MOGLS (left) and in favor of ours (right) for an RUE instance with 100 cities. The differences are not very marked, but slightly more strongly in favor of our algorithm. Note that, in order to improve visibility, the gap between the worst case and lower bound was removed since it was too tight.

Given the high performance advantage of our SLS algorithms, we decided to run some experiments comparing the SLS algorithms to MOGLS for MTSP instances with three objectives. Since the experimental study was done only for the two objectives case, we first performed some exploratory experiments concerning the configuration of the SLS algorithms. Based on these, we decided to increase strongly the number of scalarizations to 5 151 scalarizations (that is, $z = 100$), for all instance sizes. We did not apply the component-wise step for three objectives; the main reason was that the component-wise step for more than two objectives has the drawback of returning many clusters of solutions in the objective space. We therefore, compensate the lack of component-wise step by the increase on the number of scalarizations. All other components remained the same (that is, the `Restart` search strategy is applied to RUE instances of 100 cities, while `2phase` is applied for all other cases). For MOGLS, we increased

[6]The complete EAF plots are available online at `http://eden.dei.uc.pt/~paquete/mtsp`.

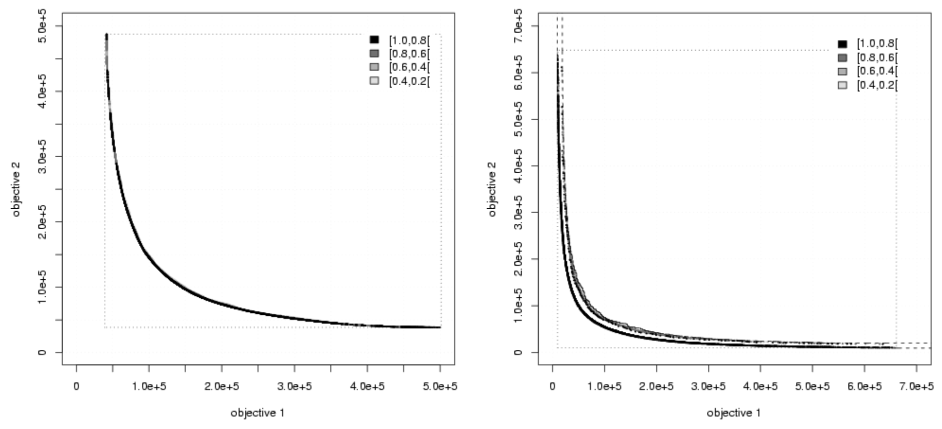

Figure 7: Location of differences between MOGLS and our SLS algorithms in favor of the latter on an RUE (left) and an RDM instance (right) with 300 cities. No differences in favor of MOGLS have been observed. Note that, in order to improve visibility, the gap between the worst case and lower bound was removed since it was too tight.

the values of the parameter l, the initial size of the archive, to 662 for instances with 100 cities, as suggested in [23], and to 1 786 for instances of size 300 by linear extrapolation.

For the instances with three objectives, we did not apply the statistical test because of the very large number of points (more than one million) that were required to define the EAFs, which anyway took already about one week for each experiment. However, the maximum absolute difference of one between the EAFs was always detected and, hence, we suspect that the null hypothesis of equal EAFs would anyway be rejected. (Recall that the statistical test is based on the maximum difference between EAFs associated to two different algorithms.) Instead, we used the parallel coordinates graphical technique [21], where each line corresponds to one point in the objective space for detecting the location of the differences. Examples of the resulting plots are given in Figure 8, where points with differences in the range of $(0.8, 1.0]$ are plotted on top and points in the range of $(0.6, 0.8]$ are plotted on bottom in favor of MOGLS (left side) and in favor of our SLS algorithms (right side). Since, for each plot a line is drawn, the much darker plots on the right side indicate a strong advantage of our SLS algorithms over MOGLS. In fact, when counting the number of points, for the RUE instance with 100 cities, there are 669 points in favor of MOGLS against 16 484 in favor of our approach in the range of $(0.8, 1.0]$ and 8,938 in favor of MOGLS against 200 618 in favor of ours in the range of $(0.6, 0.8]$. For the RDM instance of size 100, no differences above 60% were found in favor of MOGLS, and only one point was found whose difference was above 40%. Finally, for the RDM instance of size 300 only differences in favour of our approach were found.

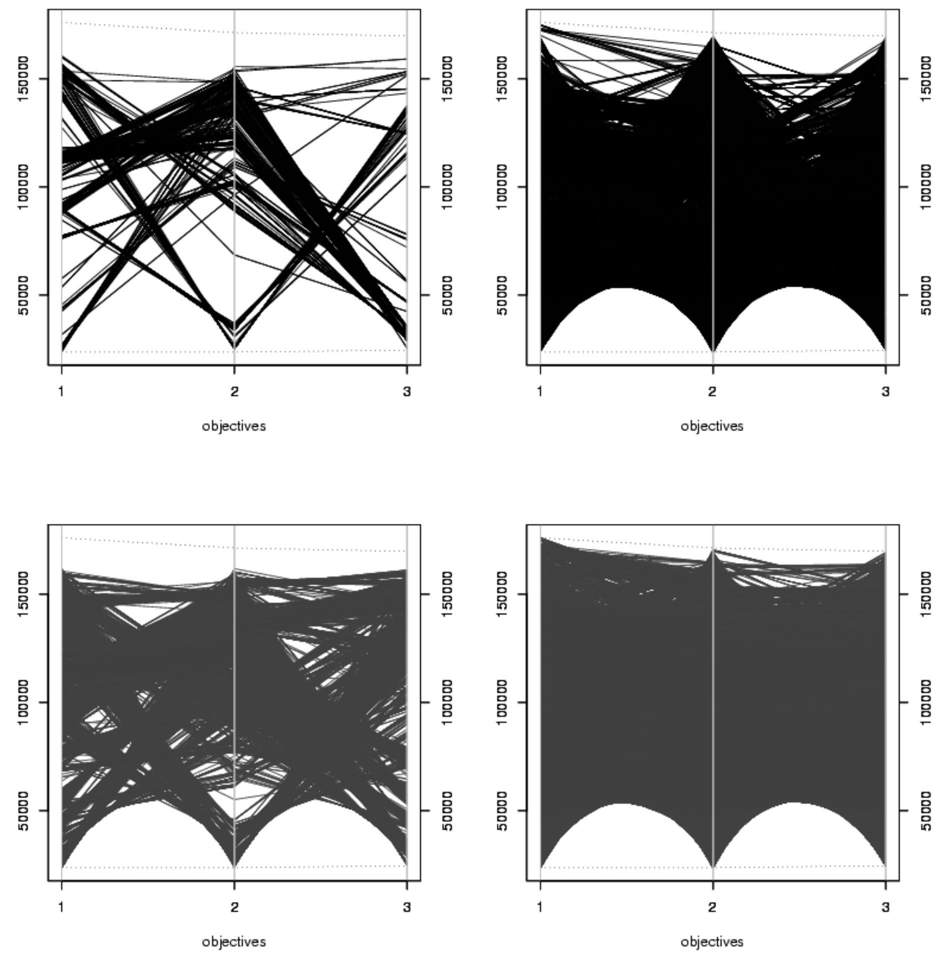

Figure 8: Location of differences between MOGLS and our SLS algorithm on an Euclidean instance of size 100 with three objectives in favor of the former (left) and in favor of the latter (right) in the range $(0.8, 1.0]$ (top) and $(0.6, 0.8]$ (bottom). See the text for more details.

Table 5 presents the average number and standard deviation of evaluations performed by our approach and by MOGLS for each instance. It is possible to observe that our approach performs much less evaluations than MOGLS. Therefore, these results indicate that our approach is highly competitive, both in terms of solution quality and time.

Type	Objectives	Size	Our approach	MOGLS
RUE	2	100	$0.04 \cdot 10^9 \pm 0.03 \cdot 10^6$	$0.10 \cdot 10^9 \pm 3.96 \cdot 10^6$
		300	$0.75 \cdot 10^9 \pm 1.29 \cdot 10^6$	$8.00 \cdot 10^9 \pm 0.85 \cdot 10^9$
	3	100	$0.41 \cdot 10^9 \pm 0.21 \cdot 10^6$	$0.69 \cdot 10^9 \pm 31.13 \cdot 10^6$
		300	$0.75 \cdot 10^9 \pm 3.52 \cdot 10^6$	$80.91 \cdot 10^9 \pm 1.36 \cdot 10^9$
RDM	2	100	$0.07 \cdot 10^9 \pm 0.26 \cdot 10^6$	$0.45 \cdot 10^9 \pm 25.33 \cdot 10^6$
		300	$0.85 \cdot 10^9 \pm 3.19 \cdot 10^6$	$57.66 \cdot 10^9 \pm 2.00 \cdot 10^9$
	3	100	$0.40 \cdot 10^9 \pm 0.26 \cdot 10^6$	$1.93 \cdot 10^9 \pm 60.92 \cdot 10^6$
		300	$1.39 \cdot 10^9 \pm 7.31 \cdot 10^6$	$287.21 \cdot 10^9 \pm 5.63 \cdot 10^9$

Table 6: The average number and standard deviation of evaluations performed by our approach and by MOGLS for each instance.

6 Discussion and Conclusions

The main goal of this paper is to make a step towards the understanding of the working mechanisms of SLS algorithms applied to the multiobjective combinatorial optimization problems from a component-based point of view. In fact, SLS algorithms are usually assembled from several components that can, or not, be instantiated for tackling a problem at hand. Hence, immediate questions for an algorithm designer are: how relevant are these components for the overall algorithm performance? Is there an component that can be removed in order to reduce the fine-tuning effort?

In this article, we focused on SLS algorithms for MCOPs that follow a general algorithmic template, the so-called Scalarized Acceptance Criterion model using their example application to the biobjective TSP. We studied the importance of their components by means of a systematic experimental design, whose results were analyzed with an sound methodology for the evaluation of the outcomes of SLS algorithms for multiobjective problems. In fact, very few studies have addressed the systematic analysis of the algorithmic components of SLS algorithms for MCOPs in a rigorous manner.

Our analysis gave clear hints on the effectiveness of each algorithmic component for the MTSP. First, strong intensification for each scalarized problem provides better performance, as shown by the results on the components neighborhood structure and search length. This gives a clear indication that further improvements could be obtained by using iterative improvement algorithms based on more advanced neighborhood structures such as used by the Lin-Kerninghan heuristic [29] and its iterated versions [3, 18, 25]. In fact, initial results by other researchers [24, 32] indicate that this conjecture may be true. The component-wise step is always recommendable, at least, for two objectives. For more objectives, it may, however, induce an undesirable clustering, which may be circumvented by using more scalarizations; nevertheless, more research in this direction is certainly necessary to give a more detailed answer. Finally, the `2phase` strategy seems to be a better option than `Restart`. This fact is certainly connected to the recent results on the *closeness* of approximate solutions for this problem [37].

As a proof-of-concept, the insights we gained from this analysis were used to assemble SLS algorithms for the MTSP. Despite their simplicity, they showed to be highly competitive with other well-established algorithms for this problem.

There are a number of possibilities for further investigations. More experimental research for this and other MCOPs is certainly required to further increase the understanding of the importance of algorithmic components of SLS algorithms. One methodological aspect that should be treated to explore other measures for comparing the efficient sets returned by the SLS algorithms such as the hypervolume indicator, the R measure, and the ϵ-indicator. Such measures would certainly speed-up computations in the analysis of the experimental results; at the same time the may incur some loss of relevant information, which is avoided by our methodology. Another direction is to analyze the importance of the algorithmic components in dependence of search space characteristics of the MCOPs and the connectedness of the solutions in the efficient set.

Ultimately, we hope that systematic experimental designs and their rigorous analysis will help to make the development of effective SLS algorithms for MCOPs less an art but more a well-established algorithm engineering process.

References

[1] E. Angel, E. Bampis, and L. Gourvés. Approximating the Pareto curve with local search for the bicriteria TSP(1,2) problem. *Theoretical Computer Science*, 310:135–146, 2004.

[2] E. Angel, E. Bampis, and L. Gourvés. A dynasearch neighborhood for the bicriteria traveling salesman problem. In X. Gandibleux, M. Sevaux, K. Sörensen, and V. T'kindt, editors, *Metaheuristics for Multiobjective Optimisation*, volume 535 of *Lecture Notes in Computer Science*, pages 153–176. Springer Verlag, Berlin, Germany, 2004.

[3] D. L. Applegate, R. E. Bixby, V. Chvátal, and W. J. Cook. *The Traveling Salesman Problem: A Computational Study.* Princeton University Press, Princeton, NJ, USA, 2006.

[4] P. Borges. CHESS – Changing Horizon Efficient Set Search: A simple principle for multiobjective optimization. *Journal of Heuristics*, 6(3):405–418, 2000.

[5] J. Conover. *Practical Nonparametric Statistics.* John Wiley & Sons, New York, NY, 1980.

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

[7] M. Ehrgott. Approximation algorithms for combinatorial multicriteria optimization problems. *International Transactions in Operational Research*, 7:5–31, 2000.

[8] M. Ehrgott and X. Gandibleux. Approximative solution methods for combinatorial multicriteria optimization. *TOP*, 12(1):1–90, 2004.

[9] V. A. Emelichev and V. A. Perepelitsa. On the cardinality of the set of alternatives in discrete many-criterion problems. *Discrete Mathematics and Applications*, 2(5):461–471, 1992.

[10] C. M. Fonseca and P. Fleming. Genetic algorithms for multiobjective optimization: Formulation, discussion and generalization. In S. Forrester, editor, *Proceedings of the Fifth International Conference on Genetic Algorithms*, pages 416–423. Morgan Kaufmann Publishers, San Mateo, CA, 1993.

[11] C. M. Fonseca and P. Fleming. On the performance assessment and comparison of stochastic multiobjective optimizers. In H. M. Voigt, W. Ebeling, I. Rechenberg, and H. P. Schwebel, editors, *Proceedings of PPSN-IV, Fourth International Conference on Parallel Problem Solving from Nature*, volume 1141 of *Lecture Notes in Computer Science*, pages 584–593. Springer Verlag, Berlin, Germany, 1996.

[12] B. Freisleben and P. Merz. A genetic local search algorithm for solving symmetric and assymetric traveling salesman problem. In T. Bäck, T. Fukuda, and Z. Michalewicz, editors, *Proceedings of the 1996 IEEE International Conference on Evolutionary Computation (ICEC 1996)*, volume 1996, pages 616–621, 1996.

[13] X. Gandibleux, N. Mezdaoui, and A. Fréville. A tabu search procedure to solve multiobjective combinatorial optimization problems. In R. Caballero, F. Ruiz, and R. Steuer, editors, *Advances in Multiple Objective and Goal Programming*, volume 455 of *Lecture Notes in Economics and Mathematics Systems*, pages 291–300. Springer Verlag, 1997.

[14] P. I. Good. *Permutation Tests: A pratical guide to resampling methods for testing hypothesis*. Springer Series in Statistics. Springer Verlag, New York, USA, 2nd edition, 2000.

[15] V. Grunert da Fonseca, C. M. Fonseca, and A. Hall. Inferential performance assessment of stochastic optimizers and the attainment function. In E. Zitzler, K. Deb, L. Thiele, C. C. Coello, and D. Corne, editors, *Evolutionary Multi-criterion Optimization (EMO 2001)*, volume 1993 of *Lecture Notes in Computer Science*, pages 213–225. Springer Verlag, Berlin, Germany, 2001.

[16] M. P. Hansen. Use of subsitute scalarizing functions to guide a local search base heuristics: The case of moTSP. *Journal of Heuristics*, 6:419–431, 2000.

[17] M. P. Hansen and A. Jaszkiewicz. Evaluating the quality of approximations to the non-dominated set. Technical Report IMM-REP-1998-7, Institute of

Mathematical Modelling, Technical University of Denmark, Lyngby, Denmark, 1998.

[18] K. Helsgaun. An effective implementation of the Lin-Kernighan traveling salesman heuristic. *European Journal of Operational Research*, 126(1):106–130, 2000.

[19] H. Hoos and T. Stützle. *Stochastic Local Search – Foundations and Applications*. Morgan Kaufmann Publishers, San Francisco, CA, 2004.

[20] J. Hsu. *Multiple Comparisons - Theory and Methods*. Chapman & Hall/CRC, 1996.

[21] A. Inselberg. The plane with parallel coordinates. *Visual Computer*, 1(4):69–91, 1985.

[22] H. Ishibuchi and T. Murata. A multi-objective genetic local search algorithm and its application to flow-shop scheduling. *IEEE Transactions on Systems, Man, and Cybernetics – Part C*, 28(3):392–403, 1998.

[23] A. Jaszkiewicz. Genetic local search for multiple objective combinatorial optimization. *European Journal of Operational Research*, 1(137):50–71, 2002.

[24] A. Jaszkiewicz and P. Zielniewicz. Pareto memetic algorithm with path relinking for bi-objective traveling salesperson problem. *European Journal of Operational Research*, In press.

[25] D. S. Johnson and L. A. McGeoch. The travelling salesman problem: A case study in local optimization. In E. H. L. Aarts and J. K. Lenstra, editors, *Local Search in Combinatorial Optimization*, pages 215–310. John Wiley & Sons, Chichester, UK, 1997.

[26] D. S. Johnson and L. A. McGeoch. Experimental analysis of heuristics for the STSP. In G. Gutin and A. Punnen, editors, *The Traveling Salesman Problem and its Variations*, pages 369–443. Kluwer Academic Publishers, Dordrecht, The Netherlands, 2002.

[27] C. P. Keller. Algorithms to solve the orienteering problem: A comparison. *European Journal of Operational Research*, 41:224–231, 1989.

[28] P. Klingsberg. A gray code for compositions. *Journal of Algorithms*, 3:41–44, 1982.

[29] S. Lin and B.W. Kernighan. An effective heuristic algorithm for the traveling salesman problem. *Operations Research*, 21(2):498–516, 1973.

[30] M. López-Ibáñez, L. Paquete, and T. Stützle. Hybrid population-based algorithms for the bi-objective quadratic assignment problem. *Journal of Mathematical Modelling and Algorithms*, 5(1):111–137, 2006.

[31] H. R. Lourenço, O. Martin, and T. Stützle. Iterated local search. In F. Glover and G. Kochenberger, editors, *Handbook of Metaheuristics*, volume 57 of *International Series in Operations Research & Management Science*, pages 321–353. Kluwer Academic Publishers, Norwell, MA, 2002.

[32] T. Lust and J. Teghem. Two phase stochastic local search algorithms for the biobjective traveling salesman problem. In E. Ridge, T. Stützle, M. Birattari, and H. H. Hoos, editors, *Proceedings of SLS-DS 2007, Doctoral Symposium on Engineering Stochastic Local Search Algorithms*, pages 21–25, Brussels, Belgium, 2007.

[33] L. Paquete. *Stochastic Local Search Algorithms for Multiobjective Combinatorial Optimization: Methodology and Analysis.* PhD thesis, Fachbereich Informatik, Technische Universität Darmstadt, 2005.

[34] L. Paquete and C. M. Fonseca. A study of examination timetabling with multiobjective evolutionary algorithms. In *Proceedings of the Fourth Metaheuristics International Conference*, pages 149–154, Porto, 2001.

[35] L. Paquete and T. Stützle. A two-phase local search for the biobjective traveling salesman problem. In C. M. Fonseca, P. Fleming, E. Zitzler, K. Deb, and L. Thiele, editors, *Proceedings of the Evolutionary Multi-criterion Optimization (EMO 2003)*, volume 2632 of *Lecture Notes in Computer Science*, pages 479–493. Springer Verlag, Berlin, Germany, 2003.

[36] L. Paquete and T. Stützle. Stochastic local search algorithms for multiobjective combinatorial optimization: A review. In T. F. Gonzalez, editor, *Handbook of Approximation Algorithms and Metaheuristics*, Computer and Information Science Series, pages 29–1—29–15. Chapman & Hall/CRC, Boca Raton, FL, USA, 2007.

[37] L. Paquete and T. Stützle. Clusters of non-dominated solutions in multiobjective combinatorial optimization. In V. Barichard, M. Ehrgott, X. Gandibleux, and V. T'Kindt, editors, *Post-Conference Proceedings of MOPGP 2006*, Lecture Notes in Economics and Mathematical Systems. Springer, 2008. In press.

[38] L. Paquete, T. Stützle, and M. López-Ibáñez. Using experimental design to analyze stochastic local search algorithms for multiobjective problems. In K. F. Doerner, M. Gendreau, P. Greistorfer, W. J. Gutjahr, R. F. Hartl, and M. Reimann, editors, *Metaheuristics — Progress in Complex Systems Optimization*, volume 39 of *Operations Research/Computer Science Interface Series*, pages 325–344, New York, USA, 2007. Springer Verlag.

[39] G. Reinelt. *The Traveling Salesman: Computational Solutions for TSP Applications*, volume 840 of *Lecture Notes in Computer Science*. Springer Verlag, Berlin, Germany, 1994.

[40] R. Ruiz and T. Stützle. A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem. *European Journal of Operational Research*, 177(3):2033–2049, 2007.

[41] P. Serafini. Some considerations about computational complexity for multi-objective combinatorial problems. In J. Jahn and W. Krabs, editors, *Recent Advances and Historical Development of Vector Optimization*, volume 294 of *Lecture Notes in Economics and Mathematical Systems*, pages 222–231. Springer Verlag, Berlin, Germany, 1986.

[42] K. J. Shaw, C. M. Fonseca, A. L. Nortcliffe, M. Thompson, J. Love, and P. J. Fleming. Assessing the performance of multiobjective genetic algorithms for optimization of a batch process scheduling problem. In *Proceedings of the 1999 Congress on Evolutionary Computation (CEC'99)*, volume 1, pages 34–75, 1999.

[43] I. K. Sigal. Algorithms for solving the two-criterion large-scale travelling salesman problem. *Computational Mathematics and Mathematical Physics*, 34(1):33–43, 1994.

[44] R. E. Steuer. *Multiple Criteria Optimization: Theory, Computation and Application.* Wiley Series in Probability and Mathematical Statistics. John Wiley & Sons, New York, NY, 1986.

[45] T. Stützle and H. H. Hoos. Analysing the run-time behaviour of iterated local search for the travelling salesman problem. In P. Hansen and C. Ribeiro, editors, *Essays and Surveys on Metaheuristics*, Operations Research/Computer Science Interfaces Series, pages 589–611. Kluwer Academic Publishers, Boston, MA, 2001.

[46] E. L. Ulungu. *Optimisation combinatoire multicritére: Détermination de l'ensemble des solutions efficaces et méthodes interactives.* PhD thesis, Université de Mons-Hainaut, Mons, Belgium, 1993.

[47] J. Xu, S.Y. Chiu, and F. Glover. Fine-tuning a tabu search algorithm with statistical tests. *International Transactions in Operational Research*, 5(4):233–244, 1998.

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