A Heuristic Selection Mechanism to Improve the Distribution of Non-dominated Fronts for the Multi-objective TSP

Nadarajen Veerapen

School of Computer Science
University of Nottingham
Jubilee Campus
Wollaton Road
Nottingham NG8 1BB – United Kingdom
nadarajen.veerapen@etu.univ-nantes.fr

Abstract. We describe a two-phase resolution method to find a good distribution of solutions along the Pareto front of the multi-objective traveling salesman problem. The first phase consists in producing a number of supported solutions. The second phase explores the front to create a better distributed set. This is done using a heuristic-selection mechanism akin to a hyperheuristic which tries to choose a good heuristic from a population of heuristics. The method can thus be applied to problems other than the TSP.

1 Introduction

Multi-objective combinatorial optimization is present in a wide array of real-life problems, for instance finance, logistics or sports (scheduling). Generally the different objectives have to be considered simultaneously. The optimal solutions are therefore ones for which there is no other solution which is better in all the objectives at the same time. Knowing the whole set of optimal solutions is often not necessary. However, having a good distribution of an approximation of this set is useful for decision makers to perform informed choices.

As resolution methods try to get better results, they often become more complicated, requiring expert knowledge to use them effectively. The hyper-heuristic approach provides a high-level view of the problem. Its aim is to allow the user to provide a number of different (simple) options to potentially solve a problem and to find on its own which options are the best. Therefore complex problems can be solved relatively easily by users who are not required to be experts in the problem domain or in the resolution method used.

The traveling salesman problem (TSP) is a classic combinatorial problem which has been extensively studied. Its objective is to find the shortest tour passing only once through every location which has to be visited. Other than transportation, this problem has applications in fields such as manufacturing (drilling of holes in printed circuit boards). It is also an important sub-problem of the vehicle routing problem.

The multi-objective version of the TSP is of interest because it represents many practical situations, for example having to make a compromise between travel time and cost of transportation.

To the best of our knowledge, hyper-heuristics have yet to be applied to the TSP or its multi-objective variant.

In this paper we propose the use of a hyper-heuristic to "populate" the Pareto front starting from a few solutions which are produced using a method which is very computationally expensive when an exact resolution is used and still remains expensive (but less so) with an approximate resolution.

In section 2 we describe the problem and provide a review of the current literature. Section 3 is devoted to the presentation of our method and it is assessed in section 4. Finally, this paper is concluded by a discussion of possible improvements and areas of future research.

2 State of the Art

In this section we present the problem and the background needed for our proposed method. We first define some terms and notions regarding multi-objective combinatorial optimization. We then look at the TSP and give an overview to heuristic search methods focusing more on hyperheuristics and low-level heuristics which may be used for the TSP. Some quality metrics are also presented.

2.1 Multi-objective Combinatorial Optimization

Combinatorial optimization involves finding the best possible solution within a set of discrete feasible solutions (Ehrgott and Gandibleux 2002). In multi-objective combinatorial optimization (MOCO), two or more (conflicting) objectives are considered. It is assumed that a solution which optimizes all objectives does not exist. The aim is to find a set of *efficient solutions*, the definition of which we give later on.

Considering p minimization objectives, a MOCO problem can be formulated as:

$$\begin{aligned} \min \quad & z(x) = Cx && (1)\\ \text{subject to} \quad & Ax = b && (2)\\ & x \in \{0,1\}^n && (3) \end{aligned}$$

where $x \in \{0,1\}^n$ are n decision variables $x_i, i \in \{1,\ldots,n\}$, $C \in \mathbb{N}^{p\times n}$ are p objective functions $C_k, k \in \{1,\ldots,p\}$, $A \in \mathbb{N}^{m\times n}$ and $b \in \mathbb{N}^{m\times 1}$ are m constraints $A_j = m_j, j \in \{1,\ldots,m\}$, to which are associated a combinatorial problem structure.

Let X be the set of all feasible solutions of the problem in the decision space, $X = \{x \in \{0,1\}^n | Ax = b\}$. Let $Z = \{z(x) | x \in X\}$ be the feasible set in the objective space.

Definition 1. Given two vectors u and v of equal cardinality p, u *dominates* v if $u_k \leq v_k, \forall k \in \{1, \ldots, p\}$ with at least one strict inequality. This is often referred to as Pareto dominance and is denoted by $u \prec v$.

Definition 2. u *strictly dominates* v if $u_k < v_k, \forall k \in \{1, \ldots, p\}$. This relationship is denoted by $u < v$.

Definition 3. u *weakly dominates* v if $u_k \leq v_k, \forall k \in \{1, \ldots, p\}$. This relationship is denoted by $u \leq v$.

Definition 4. A solution $\hat{x} \in X$ is said to be *efficient* if there is no $x \in X$ such that $x \prec \hat{x}$. The set of all the efficient solutions is denoted by X_E.

Definition 5. The image $z(\hat{x})$ of an efficient solution $\hat{x}$ is said to be a *non-dominated point*. The set of all non-dominated points is denoted by Z_N and is called the Pareto front.

Efficient solutions can be divided into two categories: supported efficient solutions and non-supported efficient solutions. Let *conv* Z be the convex hull of Z and $\mathbb{R}^p_{\geqq}$ the non-negative orthant of $\mathbb{R}^p$.

- A *supported efficient solution* is the optimal solution of a weighted sum single-objective problem with weight vector $\lambda > 0$ or, equivalently, a solution found on *conv* $Z + \mathbb{R}^p_{\geqq}$. It is said to be *extreme* if it is a vertex of *conv* $Z + \mathbb{R}^p_{\geqq}$ and *non-extreme* if it is found in its relative interior.
  The sets of supported efficient solutions and supported non-dominated points are denoted by X_{SE} and Z_{SN} respectively. The sets of extreme supported efficient solutions and extreme supported non-dominated solutions are denoted by X_{SE_1} and Z_{SN_1} respectively.
- A *non-supported efficient solution* is an efficient solution for which there does not exist an optimal solution to a weighted sum single-objective problem for any weight vector $\lambda > 0$. Non-supported efficient solutions are found in the interior of *conv* $Z + \mathbb{R}^p_{\geqq}$.

Definition 6. The *lexicographic order* $\leqq_{\text{lex}}$ between two vectors u and v is defined by $u_{k^*} < v_{k^*}$ or $u = v$ where $k^* = \min\{k | u_k \neq v_k\}$.

Definition 7. A vector $\hat{v} \in V$ is *lexicographically optimal* for some permutation π if $\hat{v}^\pi \leqq_{\text{lex}} v^\pi, \forall v \in V$.

Definition 8. A *complete* set X_{Ec} is defined as follows: $\forall z \in Z_N, \exists x \in X_{Ec}, z(x) = z$. A *minimal complete* set X_{Em} is a complete set without equivalent solutions.

In the next section we describe the traveling salesman problem which provides a basis for the case study in this paper.

2.2 The Traveling Salesman Problem

The Traveling Salesman Problem (TSP) is a well-known combinatorial optimization problem (Gutin and Punnen 2002). It represents a class of problems which are equivalent, given a list of cities and the distances between them, to finding the shortest tour passing exactly once through each city.

Its multi-objective version formalizes many practical situations with conflicting objectives. For instance, in addition to considering distance, one could also want to minimize traveled time and total cost.

Given a complete graph $G = (V, E)$ with weights $c_{ij} \in \mathbb{N}^p$ for each $(v_i, v_j) \in E, v_i, v_j \in V$, the multi-objective problem can be formulated as follows:

$$\min \quad z_k(x) = \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij}^k x_{ij}, \qquad k \in \{1, \ldots, p\} \tag{4}$$

$$\text{subject to} \quad \sum_{i=i}^{n} x_{ij} = 1, \qquad j \in \{1, \ldots, n\} \tag{5}$$

$$\sum_{j=1}^{n} x_{ij} = 1, \qquad i \in \{1, \ldots, n\} \tag{6}$$

$$\sum_{i \in S} \sum_{j \notin S} x_{ij} \geq 1 \qquad \forall S \subset \{1, \ldots, n\}, \quad S \neq \emptyset \tag{7}$$

$$x_{ij} \in \{0, 1\}, \qquad (i, j) \in \{1, \ldots, n\} \times \{1, \ldots, n\} \tag{8}$$

where all coefficients $c_{ij}^k \geq 0$ are integers and $x_{ij} = 1$ if edge (v_i, v_j) is selected (0 otherwise).

Other than a minimization objective, there are several variations of the TSP, some of which we list below (see Gutin and Punnen [2002] for an in-depth survey):

- The MAX TSP where the objective is to find a tour of maximum cost.
- The bottleneck TSP where the largest cost in the tour should be the smallest possible cost.
- The TSP with multiple visits where each node has to be visited *at least* once.
- The clustered TSP with a additional constraint stating that all the nodes in a given cluster must be visited consecutively.
- The generalized TSP which also considers clusters but where the constraint is that the tour must pass through exactly one node in each cluster.

The distances between the different nodes can be based on different norms (for example Euclidean or max) or they can just be random distances.

If the distance from any given node $v_i \in V$ to any other given node $v_j \in V$ is the same as the distance from v_j to v_i, then the TSP is said to be *symmetric* and can be referred to as STSP. Otherwise it is said to be *asymmetric* (ATSP).

The traveling salesman problem is itself a sub-problem of the vehicle routing problem (VRP). In the classic VRP, a number of identical vehicles, having a capacity constraint and starting from a central depot, are required to be optimally

routed to supply customers with known demands. Variations of this problem include vehicles with mixed capacities and customers with time windows.

The TSP and other combinatorial problems can be solved exactly or with heuristic search methods. The following sections gives an overview of such methods.

2.3 Metaheuristics

Optimization problems can either be solved exactly, for example using techniques such as branch and bound, or approximately using heuristics. This section briefly describes metaheuristics.

A metaheuristic is an approximate and often non-deterministic method used to guide the search process with the aim of efficiently exploring the search space and avoiding getting stuck in local optima. There exists a wide variety of metaheuristics (Blum and Roli 2003, Glover and Kochenberger 2003). We describe briefly a few metaheuristics which are mentioned later on in this paper.

- Simulated Annealing (SA) – The basic idea behind this approach is to allow moves which produce solutions which are worse than the current one in an attempt to escape local optima. The probability of accepting such "worsening" moves decreases as the search progresses.
- Tabu Search (TS) – This is based on a memory (which can be short-term or long-term) called the tabu list which keeps track of visited solutions. While exploring the neighborhood of a solution, the search is prohibited from using the neighbors found in the tabu list. This prevents endless loops and allows for the acceptance of potentially worse moves.
- Variable Neighborhood Search (VNS) – VNS uses a number of different neighborhoods which are explored consecutively. If there is no solution in the current neighborhood which improves the current solution, the next neighborhood is explored.
- Genetic Algorithm (GA) – Based on the concept of genes and evolution, GAs are population based. Offspring solutions are created by combining parents (often two of them). This behavior can lead to premature convergence but is countered by mutation (slight perturbations) of solutions.
- Ant Colony Optimization (ACO) – This approach models pheromone trails which ants deposit on the ground. The path with the higher level of pheromones is chosen, thus the most used path (the shortest one) gradually emerges.

After this overview of metaheuristics, we now focus on hyperheuristics in the next section. They provide a higher level of abstraction than metaheuristics.

2.4 Hyperheuristics

Hyperheuristics are a relatively recent development in search methodologies. The term *hyperheuristic* was coined by Cowling et al. [2001] to describe approaches

"at a level of abstraction above that of a metaheuristic" and having "no domain knowledge, other than that embedded in a range of simple knowledge-poor heuristics". This is motivated by the fact that modern metaheuristics tend to be complex, requiring substantial domain knowledge and a keen insight in the search method itself (for example to properly set parameters). Although these approaches have proved to be very effective, their complexity makes them expensive to implement and often domain-specific. Burke et al. [2003] argue that real-world users "are more often interested in *good enough – soon enough – cheap enough* solutions to their problems".

Hyperheuristics can be defined as heuristics which select heuristics from a population of heuristics. As such, a hyperheuristic does not operate in the *solution space* (this is done by the selected heuristics) but in the *heuristic space*. It is important to note that the heuristics can range from simple moves to more complex metaheuristics. At each step, the selection process chooses the most promising heuristic (or a combination of heuristics). Ideally, this decision should require no or minimal knowledge of how the heuristics work but rely instead on the analysis of one or more objective functions (knowing whether maximization or minimization are required) and heuristic performance indicators such as running time.

There exists a number of various hyperheuristic approaches (Chakhlevitch and Cowling 2008):

- **Random selection based** – this is the simplest approach whereby a randomly selected heuristic is applied at each decision point whether there is an improvement or not. They are usually only used in comparison to single heuristics or with more complex hyperheuristics. Variants include accepting only improving solutions, proceeding by descent or hill climbing (applying the same heuristic as long as it is producing an improvement), or accepting non-improving solution within a certain margin to avoid getting stuck in a local optimum (especially if the number of heuristics is small).
- **Greedy and peckish** – a greedy approach compares the performance of each heuristic at each decision point and selects the one producing the largest improvement (and potentially accepting a slight decrease in the objective value if no improving move is found). This behavior makes them slower than other hyperheuristics with the added disadvantage that the search space is not effectively explored. To counter the latter point, the peckish approach randomly chooses a heuristic among a candidate list of the best heuristics.
- **Metaheuristic based** – these involve using metaheuristics to select heuristics instead of solutions for which they are traditionally used for.

The first metaheuristic-like approaches were based on genetic algorithms, for example Fang et al. [1994] who called the process "evolving choice heuristics" and applied it to open shop scheduling. GA-based hyperheuristics employ indirect encoding which means that a chromosome does not contain the solution but rather the way the solution is constructed. This describes either the sequence of heuristics to be used or which heuristic to use to perform a particular move or which heuristic is associated with a particular configuration of the problem.

Simulated annealing, ant colony algorithms and variable neighborhood search have also been used to direct the exploration of heuristic space.

Tabu search has been successfully used as a heuristic selector. Kendall and Mohd Hussin [2005] proposed two versions for timetabling using 13 low-level heuristics. In the first, tabu search with hill climbing, all non-tabu heuristics are considered and the one with best improvement is applied repeatedly until no more improvement is possible, it then becomes tabu. In the second, tabu search with great deluge, a solution can be updated within a certain margin. Cowling and Chakhlevitch [2003] use a population of 95 low-level heuristics to solve a personnel scheduling problem. They use a tabu list of recently called heuristics which have not improved the solution and greedily select the best heuristic. If the latter improves the solution, it is accepted even if it is tabu (it then leaves the tabu list) otherwise the best non-improving non-tabu heuristic is applied and then becomes tabu.

Instead of comparing the heuristics in order to find the best one to apply at a given point, another technique is to learn from their past performance. One such mechanism is based on reinforcement learning (Kaelbling et al. 1996). This involves rewarding improving heuristics and punishing less successful ones at each iteration. Each heuristic is thus associated with a score which can evolve during the search process and which is used to select the most effective heuristics. The choice can be made using a roulette wheel (or fair random) mechanism or by simply selecting the heuristic with the best score (Nareyek 2003).

Another approach is the use of a choice function. Soubeiga [2003] uses a weighted sum of three functions. Two of them describe the intensification potential of the heuristic taking into consideration the change in the objective function and the amount of CPU time. The first function scores this behavior when the heuristic is applied alone and the second when it is applied in sequence with another heuristic. The third describes the diversification potential (in this case, the number of seconds since the heuristic was last called).

Hyperheuristics have also been applied to multi-objective problems. Burke et al. [2005] have applied a tabu search hyperheuristic to space allocation and timetabling (using two and three objectives respectively). They use reinforcement learning and roulette wheel selection of the objective to improve and select the best ranked heuristic. Their aim is for the non-dominated set to cover the Pareto front as widely as possible. They start with a number of random solutions. The score of each heuristic with respect to each objective is used. The results show that a single tabu list performs better than one for each objective and that their approach is competitive compared to simulated annealing.

As has been stated, hyperheuristics manipulate low-level heuristics. Next, we present some simple low-level heuristics (moves) for the TSP.

2.5 Low-level TSP heuristics

The simplest moves for the TSP are exchanges and insertions of nodes, edges or subpaths (Stattenberger et al. 2007). When discussing the way the moves are performed we consider the symmetric TSP.

- A *node insertion* involves selecting a node v_i and placing it between two consecutive nodes v_j and v_{j+1}. This is done by deleting edges (v_{i-1}, v_i), (v_i, v_{i+1}) and (v_j, v_{j+1}) and adding edges (v_{i-1}, v_{i+1}), (v_j, v_i) and (v_i, v_{j+1}).
- A subpath insertion or *k-insertion* involves selecting a subpath rooted in a node v_i consisting of k consecutive nodes and inserting it between two consecutive nodes v_j and v_{j+1}. This is done by deleting edges (v_{i-1}, v_i), (v_{i+k-1}, v_{i+k}) and (v_j, v_{j+1}) and adding edges (v_{i-1}, v_{i+1}), (v_j, v_i) and (v_{i+k-1}, v_{j+1}). A 1-insertion is a node insertion.
- A node exchange or *swap* involves selecting two nodes v_i and v_j, $i < j$, and exchanging their positions. If $j > i + 1$, this is done by removing the edges (v_{i-1}, v_i), (v_i, v_{i+1}), (v_{j-1}, v_j) and (v_j, v_{j+1}) and adding edges (v_{i-1}, v_j), (v_j, v_{i+1}), (v_{j-1}, v_i) and (v_i, v_{j+1}). If $j = i + 1$, the move is equivalent to inserting v_i between v_j and v_{j+1} (or the other way round).
- A subpath swap or *k-swap* involves selecting two subpaths consisting of k consecutive nodes, one rooted in node v_i and one in v_j, and switching them. If $j > i + k$, this is done by removing the edges (v_{i-1}, v_i), (v_{i+k-1}, v_{i+k}), (v_{j-1}, v_j) and (v_{j+k-1}, v_{j+k}) and adding edges (v_{i-1}, v_j), (v_{j+k-1}, v_{i+k}), (v_{j-1}, v_i) and (v_{i+k-1}, v_{j+k}). If $j = i + k$, the move is equivalent to doing a k-insertion.
- The k-exchange move (Lin 1965[1]) consists in dropping k non-adjacent edges and reconnecting the remaining paths using k new edges. For instance a 2-exchange involves deleting edges (v_i, v_{i+1}) and (v_j, v_{j+1}) and adding edges (v_i, v_j) and (v_{i+1}, v_{j+1}). This entails reversing one of the two subpaths remaining after dropping the two edges in order to maintain a consistent orientation.

  A local-optimum obtained through an improvement method using the k-exchange neighborhood is called k-optimal or k-opt for short. This term is often used inappropriately in the literature. It can be observed that any k-opt, $k > 2$, can be generated through a finite sequence of 2-opt moves.

The k-insertion and k-swap neighborhoods have a size of $\mathcal{O}(N^2)$ whereas the k-exchange neighborhood has a size of $\mathcal{O}(N^k)$.

The most common combination of simple moves, which aim to reduce the complexity of the 3-opt procedure, are the 2.5-opt, or $2h$-opt, (Bentley 1992[1]) and the Or-opt (Or 1976[1]). The 2.5-opt extends the 2-opt by considering a 1-insertion when the 2-opt fails to improve. The Or-opt proceeds sequentially by first considering 3-insertions then 2-insertions and finally 1-insertions. This sequential process is generalized to an arbitrary number of stages and for any ordering by Babin et al. [2005].

The 3-opt move is more effective (but more costly) than the other moves (Paquete et al. 2004). 2-opt is better than swaps or insertions. As reported by Johnson and McGeoch [1997] for single-objective optimization, 2-opt and 3-opt moves produce worse results when starting with good initial tours than when starting with tours with more defects.

[1] as described in Gutin and Punnen [2002]

The k-hyperopt move (Burke et al. 2001) requires deleting two non-overlapping sequences of k edges and finding the optimal tour of the small TSP consisting of the cities previously linked by the deleted edges and the two remaining sequences of edges (each sequence can be considered as a single edge in the small TSP).

Promising moves can be defined as moves which one expects to perform better than their original version. This can be done using neighbor lists: a list of the k closest neighbors (in ascending order) is associated to each node. The promising version of the 1-insertion would select a node v_i and insert it after a node which is one of its close neighbors. Promising moves perform better, in single-objective optimization, than their normal counterparts (Stattenberger et al. 2007).

We have already presented the TSP and the low-level heuristics which are applicable. The next subsection of our paper is devoted to heuristics for the multi-objective TSP.

2.6 Heuristics for the Multi-objective TSP

The multi-objective traveling salesman problem has been less extensively studied than its single objective variant. A review of the literature also shows that the number of objectives considered is low (we have found papers mainly concerning two objectives and some concerning three objectives). The focus on two objectives is motivated by the fact that the problem are "simpler" because of the natural ordering of efficient solutions.

Jaszkiewicz [2002] proposed a multi-objective genetic local search (MOGLS) approach which is applied to two- and three-objective TSP. An initial population is built using random solutions to each of which is applied a random weight vector. The main algorithm then creates a random weight vector u. The k best solutions of the population with respect to this vector u are selected. Two parents are drawn randomly from those k solutions, they are recombined to form a new solution which is optimized locally (using the 2-exchange neighborhood). If the resulting solution is better than the k best ones it is added to the population. The stopping criterion is a given number of iterations.

In Pareto local search (PLS) all the neighbors of every solution of the population are examined (Angel et al. 2004, Paquete et al. 2004). If a neighbor is not weakly dominated by a solution in the list of potentially efficient solutions, it is added to this list (which is also updated if the new solution dominates solutions already in the list). The neighbor is also added to the population. The algorithm ends when there are no more new solutions to examine in the population.

Pareto double two-phase local search (PD-TPLS) (Paquete and Stützle 2003), for two objectives, starts with two very good solutions (each one optimized with respect to one of the objectives and computed during phase 1). The solutions are then driven by the local search using 2- or 3-exchange neighborhoods. The local optima are added to the set representing the Pareto front. Pareto local search (2-exchange neighborhood) is then applied to this set. PD-TPLS is shown to perform better than MOGLS.

Evolutionary multi-objective simulated annealing (EMOSA) (Li and Landa-Silva 2008), applied to the biTSP, consists of two phases: the first with fixed

and the second with adaptative search directions. A random initial population is generated as well as an equal number of weight vectors Λ with uniform spread. During the first phase, for each solution x^i, a neighbor x' is generated (1-swap neighborhood). It is added to an external non-dominated population if it does not dominate x^i and replaces it with a probability dependent on temperature and λ^i. This is repeated K times for each x^i. At the end, each solution x^j in the initial population is replaced by x' if the weighted sum of the objectives is better and if λ^j is within a certain radius of λ^i (competition between solutions). The second phase consists in adjusting each λ^i with respect to x^i and its closest non-dominated neighbor in the population. The process loops back to phase 1 if the minimum temperature is not reached. Results show that EMOSA performs better than other MOSAs.

Building upon MOGLS, Jaszkiewicz and Zielniewicz [2009] present a Pareto memetic algorithm (PMA) with path relinking for the bi-objective TSP. The k number of solutions from which the parents are drawn is dependent on the number of solutions and a parameter to balance convergence and diversification. The parents are the two best among those k solutions. Solutions on the path linking the both parents are added to potentially Pareto-optimal (PP) solutions. The parents are recombined like in MOGLS. PLS is performed on PP. PMA outperforms PD-TPLS.

Paquete and Stützle [2009] investigate stochastic local search (SLS) applied to the 2- and 3-objective TSP. SLS refers to a broad category of methods and algorithms which use randomized choices in generating and modifying solutions (for example evolutionary algorithms or simulated annealing). In this instance, the previous authors consider a number of "components": search strategy, number of scalarizations (weighted sums), search length, component-wise acceptance (accepting non-dominated neighbors in the neighborhood of the result of each scalarization). Depending on the nature of instances tested, they use either a population of random initial solutions or a single one. SLS is compared to MOGLS and is found to be better.

Lust and Teghem [2009] propose a simple (no parameters) two-phase Pareto local search algorithm for the biobjective TSP (2PPLS). They show that a good initial population, the set of potentially efficient solutions which is computed in phase 1, makes PLS a very effective algorithm. The first phase is performed using Aneja and Nair's [1979] dichotomic algorithm. The second phase then explores the 2-exchange neighborhood. In addition, Lust and Teghem present a simple perturbation technique applied to the cost matrices to further improve the initial population. This step allows them to produce better results than PMA on all tests.

Measuring the quality of the solutions produced requires indicators, or metrics. We give a brief overview in the following subsection.

2.7 Quality Metrics to Assess Sets of Non-dominated Solutions

It is necessary to compare the strengths and weaknesses of different approaches to ultimately identify the most promising method. The issue of how to compare

multi-objective results is a difficult one. In single objective optimization comparing one objective against another is trivial. In comparison, the concept of quality is less clear in multi-objective optimization.

There exists a number of metrics (or performance indices) to measure the quality of a population (Okabe et al. 2003). These assess a number of aspects, namely cardinality, accuracy (how close the solutions are to the real Pareto front) and the distribution and spread. They can be based on different criteria:

- Distance – the distance between solutions in the population or between two populations.
  The R measure (ranging from 0 to 1, to be maximized) compares a non-dominated set A with the expected value of the weighted Tchebycheff utility function over a set of normalized weight vectors (Jaszkiewicz 2000).

$$R(A) = 1 - \frac{\sum_{\Lambda \in \Psi_s} s^*_\infty(z^0, A, \Lambda)}{|\Psi_s|}$$

  with $\Psi_s = \{\Lambda = [\lambda_1, \ldots, \lambda_J] | \sum_{j=1}^{J} \lambda_j = 1, \lambda_j \in \{0, 1/k, 2/k, \ldots, (k-1)/k, 1\}\}$ and $s^*_\infty(z^0, A, \Lambda) = min_{z \in A}\{max_j\{\lambda_j(z_j - z^0_j)\}\}$ where k is a sampling parameter, J the number of objectives and z^0 the ideal point. The objective values are normalized.
- Volume – the volume of the area dominated by the solutions is considered,
  One such metric is the hypervolume indicator H (Zitzler and Thiele 1999). It indicates the volume of the objective space dominated by the points in the solution set A and dominating a reference point y^* (the higher the better).

$$H(A, y^*) = \Lambda(\cup_{u \in A}\{y | u \prec y \prec y^*\})$$

  where Λ is the Lebesgue measure of a set.
  The hypervolume is strictly Pareto compliant (Zitzler et al. 2003), which is to say that given two Pareto sets A and B the indicator value of A will be higher than that of B if A dominates B. Hypervolume favors the convex parts of the front.
- Niching – this takes into account the number of solutions within a certain radius of given points. The $\mathcal{M}^*_2$ index (Zitzler et al. 2000) is one example of a performance index which assesses distribution and takes into account the number of non-dominated solutions. It reflects the number of niches in A (the higher the better).

$$\mathcal{M}^*_2 = \frac{1}{|A| - 1} \sum_{a_1 \in A} |\{a_2 \in A | \|a_1 - a_2\| > \sigma\}|$$

  where σ is the niche radius.
- Entropy – this is based on Shannon's entropy.
  Farhang-Mehr and Azarm [2003] have proposed one such metric. Each solution point provides some information about its neighborhood: the influence function (it is suggested that this be modeled by a Gaussian function). The

aggregation of the influence functions of all solution points is the density function. A good distribution of solutions would produce a density function with a relatively even surface.

The number of quality indicators used to compare results should be at least the same as the number of objectives to be able to detect whether one objective vector dominates another (Zitzler et al. 2003).

This concludes the first part of our paper. The next section deals with our proposed method.

3 The Proposed 2-Phase Approach

Our algorithm consists of two phases. The first one computes a very good approximation of the set of supported efficient solutions using the same method employed by Lust and Teghem [2009] for the bi-objective problem. When considering problems with more objectives, in our case three-objectives, the first phase uses the algorithm proposed by Przybylski et al. [2009].

The second phase iteratively drives a subset of the population across the potential Pareto-front with the goal of maximizing the hypervolume. The non-dominated solutions explored are added to the population.

Gandibleux et al. [2003] use the same idea of paying a certain computational price for good initial solutions which then help the second phase heuristics in their search.

3.1 Phase 1

The first phase works by intensively using well-known efficient algorithms for mono-objective combinatorial optimization. This involves preserving the structure of the problem at all times with the weighted sum being the only usable scalarization.

Two objectives Phase 1 is based on the dichotomic algorithm by Aneja and Nair [1979] which determines a minimal complete set of supported efficient solutions. Using weighted sum scalarization, we solve $\min_{x \in X}\{\lambda_1 z_1(x) + \lambda_2 z_2(x)\}$ where $\lambda_1, \lambda_2 > 0$.

Given two supported solutions x^1 and x^2 with $y^1 = z(x^1)$ and $y^2 = z(x^2)$ such that $y_1^1 < y_1^2$ and $y_2^1 > y_2^2$, let $\lambda = (\lambda_1, \lambda_2)$ be the weight defined by $\lambda_1 = y_2^1 - y_2^2$ and $\lambda_2 = y_1^2 - y_1^1$. λ defines the normal to the line connecting y^1 and y^2, $\lambda_1 y_1^1 + \lambda_2 y_2^1 = \lambda_1 y_1^2 + \lambda_2 y_2^2$.

Let $\hat{x}$ be an optimal solution of the weighted sum scalarization, with $\hat{y} = z(\hat{x})$. Suppose $\lambda_1 \hat{y}_1 + \lambda_2 \hat{y}_2 < \lambda_1 y_1^2 + \lambda_2 y_2^2$, then $\hat{y}$ is located below the line connecting y^1 and y^2.

We then recursively check to see if there are any supported solutions between y^1 and $\hat{y}$ and between $\hat{y}$ and y^2.

Algorithm 1: BiobjectiveDichotomy

Compute $x^{(1,2)}$ and $x^{(2,1)}$ two lexicographically optimal solutions for $z^{(1,2)}$ and $z^{(2,1)}$ respectively ;
$\tilde{X} \leftarrow \{x^{(1,2)}, x^{(2,1)}\}$;
$\tilde{X} \leftarrow$ `SolveRecursion(`$x^{(1,2)}$, $x^{(2,1)}$, $\tilde{X}$`)` ;

Algorithm 2: SolveRecursion

Input: $x^1 \in X_{SE}$, $x^2 \in X_{SE}$, $\tilde{X} \subseteq X_{SE}$
Output: $\tilde{X} \subseteq X_{SE}$
$\lambda_1 \leftarrow z_2(x^1) - z_2(x^2)$;
$\lambda_2 \leftarrow z_1(x^2) - z_1(x^1)$;
$x \leftarrow$ `SolveWeightedSum(`λ`)` ;
$\tilde{X} \leftarrow \tilde{X} \cup \{x\}$;
if $\lambda_1 z_1(x) + \lambda_2 z_2(x) < \lambda_1 z_1(x^1) + \lambda_2 z_2(x^1)$ **then**
    $\tilde{X} \leftarrow$ `SolveRecursion(`x^1, x, $\tilde{X}$`)` ;
    $\tilde{X} \leftarrow$ `SolveRecursion(`x, x^2, $\tilde{X}$`)` ;
end

If $\lambda_1 \hat{y}_1 + \lambda_2 \hat{y}_2 = \lambda_1 y_1^2 + \lambda_2 y_2^2$, we can get a non extreme solution or a point equivalent to y^1 or y^2.

The algorithm requires two lexicographically optimal solutions to start (Algorithm 1). Let $\tilde{X}$ be the set of efficient solutions computed by the algorithm.

At the end of this method, $\tilde{X}$ contains a minimal complete set of supported efficient solutions plus possibly some non-extreme or equivalent supported solutions. A complete set of supported solutions is not necessarily obtained.

Three objectives and more It is not easy to generalize the dichotomic scheme to problems with more than 2 objectives because it relies on the natural ordering of non-dominated points in the objective space, that is $z_1(x) < z_1(y) \Rightarrow z_2(x) > z_2(y)$. A straight-forward extension of the method of Aneja and Nair [1979] cannot determine the set of extreme supported points, examples can be found in Przybylski et al. [2009].

Przybylski et al. [2009] propose a generalization by defining the adjacency of points involving weight space decomposition. Let (P_λ) define a weighted sum problem. Based on the following observations:

- all optimal solutions of (P_λ) with $\lambda \in \mathbb{R}^p_>$ are efficient,
- for $\alpha \in \mathbb{R}_>$, (P_λ) and $(P_{\alpha\lambda})$ have the same set of optimal solutions, and
- the weight set Λ can be interpreted as a set of equivalence classes

the previous authors show that geometric duality can be used to identify faces of the convex hull in the objective space by using suitable functions on a polytope included in $\mathbb{R}^p_>$.

The algorithm (Algorithm 3) starts with a suitable subset of non-dominated extreme points $\{y_1, \ldots, y_k\}$. It computes the subsets $W_0(y_i)$ of $W_0 = \{\lambda \in$

Algorithm 3: Phase One

Input: A polytope H, the cost functions Z
Output: The set S of local non-dominated extreme points with respect to H
if *dim* $H == 2$ **then** `biObjectiveDichotomy(`Z, S`)`;
else
  if H *is the initial polytope* **then** Compute the lexicographically optimal point for each permutation of objectives and add it to S ;
  else Compute one optimal solution for P_λ for each extreme point λ of H and add it to S;
end
Create empty adjacency lists $A(y)$ for each point y of S;
$i \leftarrow 1$;
while $i \neq length(S) + 1$ **do**
  Compute the polytope $H_p(y^i)$ using S;
  while $H_p(y^i) \neq H(y^i)$ *and* $dimH_p(y^i) == H(y^i)$ **do**
    Choose a facet F of $H_p(y^i)$ defined by $\langle \lambda, y^i \rangle = \langle \lambda, y^* \rangle, y^* \notin A(y^i)$;
    Compute the set S_F of local non-dominated extreme points with respect to F: `phaseOne(`Z, F`)`;
    Identify the definitive face and add new points to S;
  end
  $i \leftarrow i + 1$;
end

$\mathbb{R}^p | \sum_{k=1}^{p} \lambda_k = 1\}$ for which y_i attains minimal values of $\langle \lambda, y \rangle$ over Z. This is done by computing the boundary of each set $W_0(y_i)$, and allows the discovery of new non-dominated extreme points y. The procedure stops when $W_0(y)$ does not change for any y, at which stage a complete set Z_{SN_1} is known. The algorithm does not only compute all non-dominated extreme points but also the corresponding partition of the weight set W_0 . This fact is used to determine appropriate weight vectors to obtain the set Z_{SN} (with the maximal complete set X_{SEm}) by enumeration as well as the faces of *conv* Z defined by the non-dominated extreme points.

Our initial approach involved using an exact solver, Concorde, to solve the mono-objective problem (`solveWeightedSum`). Considering the running time and numerical instability which appears for instances with 100 cities or more (thus not giving exact results), we use the very effective Lin-Kernighan heuristic (Lin and Kernighan 1973), an implementation of which is also provided by Concorde.

Now that phase 1 has been described, the following part of this paper deals with phase 2 which uses the solutions produced in phase 1.

3.2 Phase 2

Using the set obtained during the first phase, the second phase's aim is to further explore the Pareto front without needing extensive knowledge of the problem being tackled. The pseudocode is provided in algorithm 5.

Function `update`

Input: A set of points $X \updownarrow$, a new point $p \downarrow$
Output: A boolean telling if p has been inserted in X
forall $x \in X$ **do**
    if $z(x) \leq z(p)$ **then return** false;
    if $z(p) \prec z(x)$ **then** $X \leftarrow X \backslash \{x\}$;
end
$X \leftarrow X \cup \{p\}$;
return true;

The algorithm requires a set H of one or more heuristics whose only requirement is to implement a simple interface so that the search mechanism can manipulate them. A heuristic needs to be able to generate the neighbors of a given solution point, move between the neighbors, evaluate the quality a point and apply the move should the main algorithm require it.

The algorithm needs an initial population P_i, $|P_i| \geq 1$, it maintains an archive A of all non-dominated points found and, at each iteration, only uses a running population P of maximum size S. The use of the non-dominated archive can be seen as a cut-down version of Pareto local search. The objective being to explore the front, we use the hypervolume metric to direct the search (by trying to maximize it). This allows us to consider the movement of each solution with respect to the current running population and not as a point on its own.

At each iteration, each solution $p \in P$ is considered, a heuristic is selected and the neighborhood of p is explored to find a new point which increases the hypervolume. Any non-dominated solution found during this process is added to the archive. Should an improving solution be found, the same heuristic is applied to the new point which has just been generated in a descent fashion.

If the maximum size of the running population is not reached, improving solutions are added to P as is, otherwise they replace the point they were a neighbor of. If no solutions in P were moved during the last iteration, a new running population is randomly selected from the archive. This prevents the algorithm from getting stuck in a local optimum and contributes to the diversity of the solution set.

Since calculating the hypervolume and non-dominated sorting occur constantly throughout the algorithm, it is better to keep S small. We arbitrarily choose $S = 20$. We use the hypervolume code from Fonseca et al. [2006].

Heuristic selection mechanism We wish to evaluate and use the best performing heuristics. At the start of the search, all heuristics have the same score, $r(h)$.The performance of the heuristics is inferred through a system of reward and punishment, whereby improving heuristics obtain a higher score ($r(h) \leftarrow r(h)+\alpha$) and non-performing heuristics a lower one ($r(h) \leftarrow r(h)+\gamma, \gamma < 0$). This is inspired by the principles of reinforcement learning (Kaelbling et al. 1996).

Algorithm 5: HHSolver

```
Input: A population of heuristics H, an initial population of solutions P_i, a
       running population size S, the cost matrices of the problem
Output: The final population P
T ← ∅; A ← P_i; m ← 0;
while stopping condition not reached do
    if m == 0 then
        if |A| ≤ S then P ← A;
        else P ← {S randomly selected solutions from A};
    v ← hyperVolume(P);
    m ← 0;
    forall p ∈ P do
        descent ← false;
        o ← chooseObjective(p);
        h ← chooseHeuristic(H, p, o);
        repeat
            N ← neighbors of p in h neighborhood;
            v_n ← 0;
            while N ≠ ∅ and v_n ≤ v and time condition do
                Select and remove a neighbor n from N;
                if update(A, n) then
                    P_t ← P\{p};
                    update(P_t, n);
                    v_n ← hyperVolume(P_t);
            if v_n > v then
                if descent == false then
                    r(h) ← r(h) + α;
                    m ← m + 1;
                else r(h) ← r(h) + β;
                descent ← true;
                v ← v_n;
                P ← P ∪ {n};
                if |P| > S then P ← P\{p};
                p ← n;
                empty T;
            else
                r(h) ← r(h) + γ;
                descent ← false;
                if |T| + 1 = |H| then empty T;
                else T ← T ∪ {h};
        until not descent ;
```

To supplement this strategy, a tabu list is also used to prevent worse heuristics from being used during a certain amount of time (even if it has been performing well previously). A heuristic is included in the tabu list if it has not been able to improve the distribution by moving a solution in the given amount of time it was

allowed to run. However, if an improving solution has been found, the tabu list is cleared (it is also cleared if all heuristics turn out to be tabu). The tabu list is thus of variable length. No aspiration criterion is used. Burke et al. [2003] point out that, given the multiobjective nature of the problem, it may be interesting to consider one tabu list for each objective. We choose not to since the results of the previous authors do not show any benefit of proceeding in this way. In addition we remark that we are in reality using only one objective, hypervolume. Also, our test problem's nature is simple with 2 or 3 objectives of the same nature, namely minimisations of the sum of the edge costs.

However, we also take into consideration the fact that certain heuristics may need to know in which direction (with regard to which objective) the search needs to be directed. For instance, the simple 1-insertion for the TSP does not need this information but the promising version does. This is because the latter does not use random selection of the insertion position but iterates through the list of closest neighbors of the city which is to be inserted. Thus we need to specify with respect to which objective this notion of closeness is defined. This is done by roulette wheel selection with the probability of an objective being selected being inversely proportional to its quality (the worst objective has more chance of being selected).

Two heuristic-selection mechanisms are implemented:

- Best rank – the heuristic with the best rank is selected. Ties are broken by randomly selecting, with equal probability, a heuristic from the ones with the same score. The initial score of each heuristic when using this approach is 1, $r(h) \leftarrow 1$. There is no minimal score.
- Roulette wheel – the heuristic is selected randomly with a chance proportional to its score. The initial and minimal score of heuristics in this case is 1, $\min r(h) = 1$.

We choose a simple reward and punishment scheme with $\alpha = 1$ and $\gamma = -0.5$ (the algorithm is often faced with non-improving moves). If descent occurs, that is the heuristic is applied repetitively to a point and its successive improving solutions, the reward is decreased, $\beta = 0.2$. This is motivated by the fact that we wish to reward improvement of solutions throughout the population and not on one solution in particular.

Having described our proposed approach, the next section is devoted to evaluating its performance by comparing different variants of the approach and also includes comparisons with existing methods.

4 Performance Assessment

4.1 Methodology

We test our algorithm on a number of instances to evaluate its performance:

– One biobjective instance with 100 cities with Euclidean distances, kroAB100, created by combining the single objective (kroA100 and kroB100) instances of TSPLIB[2].
– One 100-city clustered instance from DIMACS Challenge[3]. The instance is created such that the location of the cities are in the same range as the *kro* 100-city instances. The number of clusters is $N/25$, that is 4.
– One 200-city instance, kroAB200.
– One three-objective instance with 50 cities, kroABC50, based on the leading 50 cities of the respective *kro* instances.

First we test phase 1 to determine the number of solutions produced and the time taken to do so and experiment with allowing the process to complete successfully and also stopping it after a number of recursions. This provides us with a number of populations to use in phase 2:

– The MIN population consists of k solutions for the k-objective problem. Each solution is optimized considering one of the objectives.
– The DX population consists of the solutions of the phase 1 when it is restricted to a depth of X recursive calls of `biObjectiveDichotomy`. Restricting the number of calls reduces the number of solutions produced but they are still distributed all along the front, at least on the convex fronts presented here. We only consider D5 and D7.
– The ALL population consists of all the solutions produced during phase 1.

For the second phase, the following 11 low-level heuristics are employed:

– 1-, 2- and 3-insertion, as well as a promising 1-insertion (1INS, 2INS, 3INS and P1INS).
– 1-, 2- and 3-swap, as well as a promising 1-swap (1SWP, 2SWP, 3SWP and P1SWP).
– 2-exchange as well as a promising version (2EXC and P2EXC).
– A dummy heuristic which only inverts the first and second city in the tour (DUM). The purpose of this move is to see if the heuristic-selection mechanism functions correctly, that is uses it the least.

We first compare different versions of our algorithm to see if one performs better than the rest: random heuristic selection (RND), best-ranked heuristic with tabu list (BRTL), roulette wheel (RW), roulette wheel with tabu list (RWTL). We also test the algorithm using only one low-level heuristic. This is done twice, with the 2EXC and P1INS single moves which are the best performing heuristics in our tests.

We then compare our results to those of EMOSA (Li and Landa-Silva 2008) and 2PPLS (Lust and Teghem 2009) for the instances they have published results for. We have also implemented our own version of 2PPLS to be able to do comparisons with other instances.

[2] The data can be obtained at http://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/)
[3] http://www.research.att.com/~dsj/chtsp/

The metrics used are selected from the ones already presented based on the ones for which results are available. We choose the metrics so that they can be applied to more than two objectives:

- The hypervolume $\mathcal{H}$. With the reference point being $(180000, 18000)$ for the comparison of the 100-city instances and $(370000, 370000)$ for 200 cities with the results of Lust and Teghem [2009]. Otherwise it is set as the maximum value of each objective in the solution population. We record two hypervolume measures: $R\mathcal{H}$ which is the average of the hypervolume of the running population of 20 solutions at the end of each iteration and $F\mathcal{H}$ which is the hypervolume of the final population of potentially efficient solutions.
- The R metric. The normalization is carried out considering the ideal point and the reference point for the hypervolume. In keeping with the literature, the sampling parameter k is set to 100 for the bi-objective problems and to 40 for the ones with three objectives.

We also record the number of potentially efficient solutions, $|PE|$. We recall that at least as many quality indicators as objectives are needed to really determine if one result is better than another and we note that we only use two metrics for the three-objective case.

The average of the results for 10 runs are used.

Experiments are run on a PC 2.26GHz Core 2 Duo with 2GB RAM under Ubuntu 8.10, Linux kernel 2.6.27, g++ 4.3.2 with `-O2` flag, single-threaded.

4.2 Experimental Results

Table 1 shows the average number of solutions produced and the time taken by the exact resolution (EXA-ALL) and Lin-Kernighan (LK-XX) heuristic for the four test instances. While this is not reflected by the averages in the table, there is some numerical instability for the exact resolution method with the number of solutions produced varying slightly between different runs. It can be observed that the heuristic method greatly reduces the execution time of phase 1 and makes it acceptable when we consider that phase 2 is allowed to run for 30 seconds.

Tables 2, 3, 4 and 5 show the results comparing the different versions of the phase 2 algorithm when allowed to run for 30 seconds. All 10 runs of each algorithm start with the same initial population. The tables present how the starting solution set was obtained and its size, the algorithm used, the number of iterations performed within 30 seconds, the results for the quality metrics and the number of potentially efficient solutions.

It is immediately clear that none of the three selection mechanisms (BRTL, RW and RWTL) performs very well all round. Indeed, the best performer is the P1INS move when it is alone. This is easily explained by the fact that it has the smallest neighborhood (for each city only its 20 closest neighbors are considered) which is created to be of very good quality. This allows P1INS to outperform the other methods because it is able to consistently run many more

Table 1. Phase 1 results

Instance	Stop criterion	Solutions	Time(s)
KroAB100	EXA-ALL	110.0	403
	LK-ALL	110.7	19
	LK-D7	97.0	7
	LK-D5	33.0	2
	LK-MIN	2.0	~ 0
Clustered100	EXA-ALL	109.0	124
	LK-ALL	111.0	47
	LK-D7	85.8	18
	LK-D5	32.4	5
	LK-MIN	2.0	~ 0
KroAB200	EXA-ALL	222.0	1653
	LK-D7	122.8	18
	LK-D5	33.0	4
	LK-MIN	2.0	~ 0
KroABC50	EXA-ALL	286.3	450
	LK-ALL	295.3	103
	LK-D5	285.2	58
	LK-MIN	3.0	~ 0

Table 2. Phase 2: KroAB100 results

Start	Algorithm	Iterations	R $\mathcal{H}(10^8)$	F $\mathcal{H}(10^8)$	R	$\lvert PE\rvert$
All (110)	BRTL	*62.0*	211.13	217.82	0.935156	805.2
	RW	46.8	211.00	217.80	**0.935188**	733.2
	RWTL	45.2	**211.48**	217.80	0.935133	730.3
	RND	46.6	211.37	217.78	0.935136	687.6
	2EXC	34.5	*211.40*	*217.83*	0.935139	*817.2*
	P1INS	**218.2**	211.15	**217.96**	*0.935177*	**1593.8**
D5 (33)	BRTL	*40.2*	**212.92**	**217.70**	*0.935149*	847.5
	RW	39.3	212.53	217.63	0.935092	769.2
	RWTL	39.4	*212.90*	217.62	0.935132	773.9
	RND	39.0	212.02	217.57	0.935146	742.7
	2EXC	32.0	212.71	*217.69*	**0.935220**	*856.2*
	P1INS	**204.2**	211.44	**217.70**	0.934949	**1496.9**
Min (2)	BRTL	119.8	158.28	209.81	0.929568	382.3
	RW	73.0	176.16	211.35	**0.931365**	520.0
	RWTL	65.5	*178.30*	*211.98*	0.931074	542.1
	RND	44.4	**184.09**	211.56	0.930923	494.9
	2EXC	*152.0*	168.93	**212.26**	*0.931314*	*571.9*
	P1INS	**187.9**	171.80	205.49	0.922950	**971.4**

Table 3. Phase 2: Clustered100 results

Start	Algorithm	Iterations	R $\mathcal{H}(10^8)$	F $\mathcal{H}(10^8)$	R	$\|PE\|$
D5 (31)	BRTL	*38.5*	210.10	214.44	*0.946663*	*905.2*
	RW	37.7	*210.35*	214.46	**0.946676**	828.1
	RWTL	38.2	**210.56**	*214.46*	0.946660	827.7
	RND	38.4	209.86	214.36	0.946634	809.5
	2EXC	31.4	209.96	214.44	0.946611	899.6
	P1INS	**174.6**	208.83	**214.56**	0.946662	**1652.0**
D7 (89)	BRTL	*40.3*	**208.06**	*212.83*	0.946658	*832.8*
	RW	39.9	*207.97*	212.81	0.946630	779.5
	RWTL	39.6	207.47	212.79	0.946606	733.1
	RND	39.6	207.57	212.77	0.946627	707.9
	2EXC	32.1	207.39	*212.83*	*0.946692*	831.9
	P1INS	**197.2**	207.12	**212.97**	**0.946730**	**1650.5**

Table 4. Phase 2: KroAB200 results

Start	Algorithm	Iterations	R $\mathcal{H}(10^8)$	F $\mathcal{H}(10^8)$	R	$\|PE\|$
D5 (33)	BRTL	35.0	833.19	**851.65**	0.874866	*1098.3*
	RW	*35.7*	**835.98**	851.46	0.874853	1032.9
	RWTL	34.8	834.36	851.46	*0.874881*	992.5
	RND	33.6	835.21	851.34	0.874764	927.8
	2EXC	27.5	*835.78*	*851.61*	**0.874934**	898.0
	P1INS	**52.9**	825.87	851.25	0.874434	**1481.3**
D7 (124)	BRTL	40.4	828.95	853.42	0.875374	*1018.7*
	RW	*40.8*	831.03	853.35	*0.875403*	919.1
	RWTL	37.7	*831.35*	*853.43*	0.875376	920.3
	RND	36.8	831.12	853.39	0.875334	880.0
	2EXC	28.9	**831.72**	853.35	0.875340	749.5
	P1INS	**74.2**	828.62	**853.65**	**0.875460**	**1616.9**

Table 5. Phase 2: KroABC50 results

Start	Algorithm	Iterations	R $\mathcal{H}(10^{12})$	F $\mathcal{H}(10^{12})$	R	$\|PE\|$
D5 (275)	BRTL	34.3	**255.48**	**308.59**	*0.982780*	2443.4
	RW	34.8	*254.61*	*306.92*	0.982761	3998.7
	RWTL	*40.0*	252.87	299.22	**0.982817**	4644.9
	RND	36.7	252.66	299.26	0.982751	3821.3
	2EXC	33.3	253.70	305.41	0.982773	*4283.1*
	P1INS	**48.2**	245.35	300.88	0.982734	**4714.0**
Min (3)	BRTL	75.8	161.69	232.64	*0.981292*	3225.8
	RW	63.3	180.35	**237.46**	0.981236	3313.2
	RWTL	53.5	*180.78*	234.52	0.981256	3538.5
	RND	45.3	**196.67**	228.92	0.981267	3246.2
	2EXC	*99.3*	164.98	*237.07*	**0.981404**	*3523.5*
	P1INS	**100.5**	161.57	230.36	0.978900	**3642.6**

iterations within the same time frame. It is therefore able to create a much larger population which influences very favorably the quality of the metrics such as the final hypervolume, $F\mathcal{H}$, or the R measure. This behavior can be seen as a cut-down version of PLS. We note however that it is not very good for the hypervolume of the running population. In contrast the heuristic-selection mechanisms perform much better for $R\mathcal{H}$ which is to be expected since this is what is driving the search.

As expected, RND selection is outperformed by other selection mechanisms in general.

Setting aside P1INS, the selection mechanism which seems to perform the best is BRTL. It is apparent that the best performance from this algorithm comes when it starts off with a solution of "moderate" quality. If the starting population is very small (MIN), the running population quickly locates the good solutions to maximize hypervolume. It then seems to get stuck in a local optimum because when the process is restarted the random selection of a new population can only choose from a limited part of the front. If the starting population is too good (ALL and sometimes D7), there is a very small margin of improvement and the gains in hypervolume are mainly due to the addition of non-dominated solutions to the final non-dominated set. In contrast with an average starting population, the search is not biased towards the more PLS-like P1INS and the selection mechanism has a population which is diverse enough that when a new population is randomly selected, this choice can be performed all along the front.

Visual examination of the solutions sets obtained when starting with MIN (Figure 1), reveals that the large majority of the points are found at the center (most curved part) of the front and not many near the extremities. This seems to be an effect of the bias of the hypervolume metric in favor of convex fronts.

Figures 2 and 3 show the typical behavior of low-level heuristic selection for BRTL. Good moves get progressively selected more often. Bad moves such as DUM still get selected because of the tabu list feature. We stress that this behavior is by design since the heuristic selection mechanism does not know that DUM will always fail. When other, better, heuristics fail, it still tries the bad ones in an effort to improve the situation.

Table 6 presents the results of the comparison between BRTL (starting with the solutions produced without prematurely ending phase 1) and EMOSA for the instances published by Li and Landa-Silva [2008].

BRTL manages to outperform EMOSA for the hypervolume metric in three out of four instances.

Table 7 gives the results[4] of the comparison between BRTL and 2PPLS. Both BRTL and 2PPLS are given the same initial population of a full phase 1 execution. It is clear that 2PPLS outperforms BRTL although they are relatively close on the hypervolume and R metrics for the 100-city instances (but 2PPLS is much faster and produces more potentially efficient solutions). This is confirmed visually in Figure 1 when both approaches are compared.

[4] Recall that the reference point for the computation of hypervolume here is the same as in Lust and Teghem [2009] and not the one used in the previous tables.

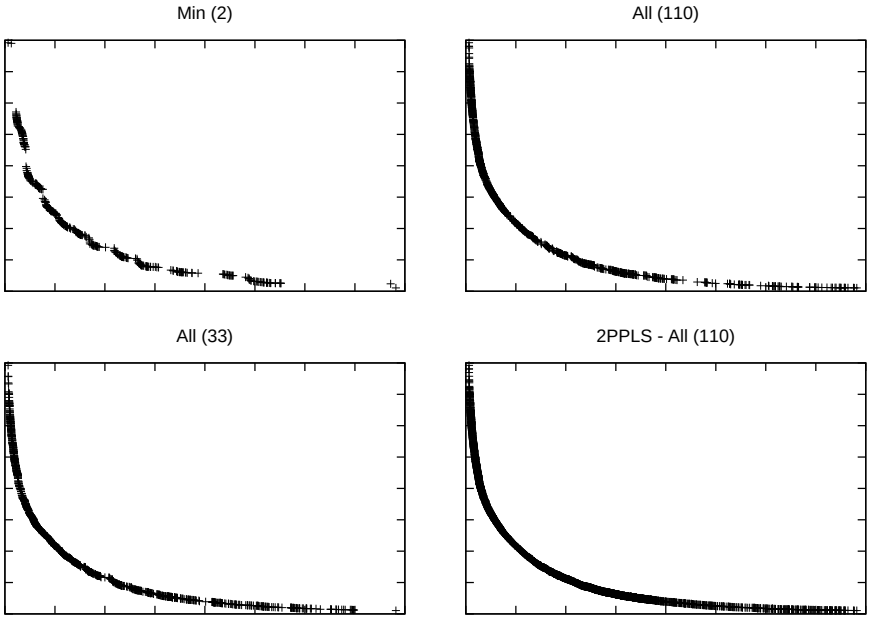

Fig. 1. Sample Pareto fronts for our approach (BRTL) starting with Min(2), D5(33) and All(110) solution sets and 2PPLS starting with All(110).

Table 6. Phase 2: Hypervolume (10^{10}) comparison with EMOSA

Instance	BRTL	EMOSA
kroAB50	**0.3544**	0.2839
kroBC50	**0.4327**	0.2809
kroAB100	**2.1782**	1.9060
kroBC100	1.8630	**1.9392**

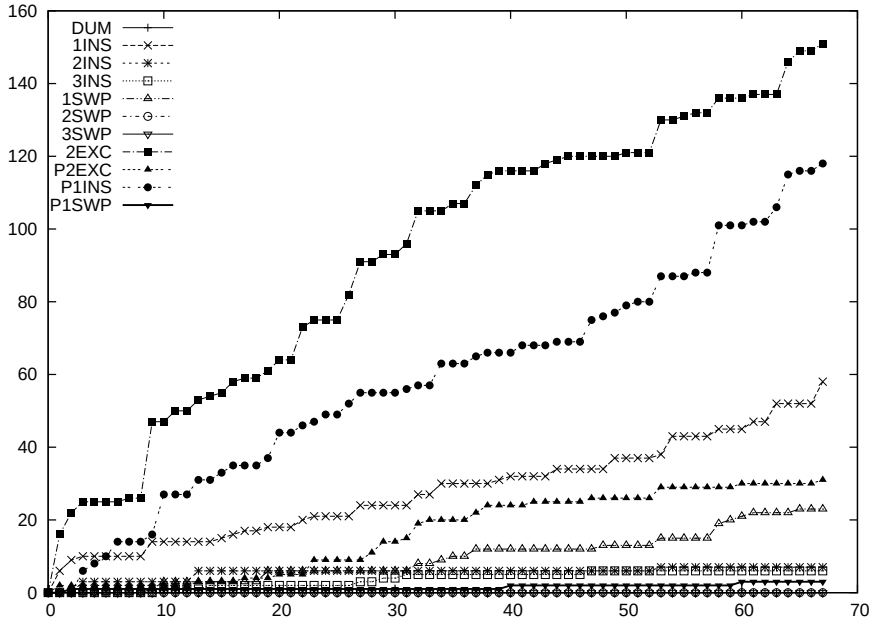

Fig. 2. Cumulative number of improving moves for each low-level heuristic against the number of iterations for one run of BRTL using kroAB100. The 2EXC and P1INS are the two best moves in this case.

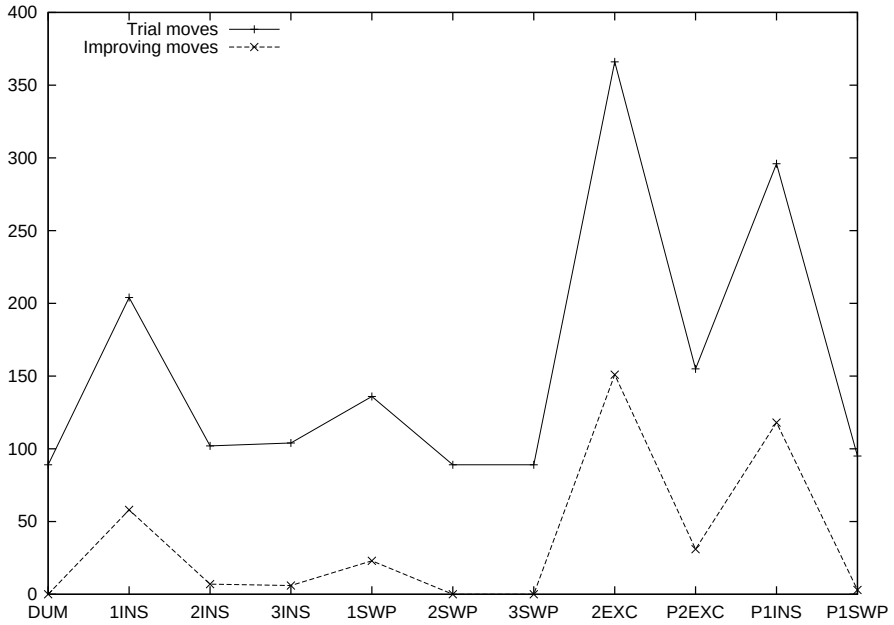

Fig. 3. Total number of trial and improving moves for each low-level heuristic for one run of BRTL using kroAB100.

We stopped the three-objective instance running on 2PPLS before it was completed because it was taking more than an hour to run. There is a second set of results for 2PPLS with three objectives: one which is stopped after 30 seconds (the same time given to BRTL). For the same running time of 30 seconds, BRTL performs better than 2PPLS.

5 Conclusion

In this paper we have presented a new heuristic-selection mechanism. We have shown through a number of experiments that it performs moderately well for average starting populations but is clearly outperformed for very small or already very well distributed sets. Although it is able to perform better than some previous approaches, it lags being 2PPLS. 2PPLS is an extreme efficient algorithm both in terms of its simplicity and the quality of its results.

There are a number of potential paths for further investigation:

- Our algorithm is coded such that it can be relatively easily modified to tackle another problem. It would therefore be interesting to check if it performs any better on other, more complicated, problems.
- The use of Concorde as our single-objective solver precludes the use of objectives other than a minimization of the sum of the edge costs. It would

Table 7. Phase 2: Comparison with second phase of 2PPLS

Instance	Algorithm	$\mathcal{H}(10^8)$	R	$\|PE\|$	Time(s)
kroAB100	BRTL	225.84	0.935156	805.2	30
	2PPLS	**226.11**	**0.935259**	**2541.7**	**13**
Clustered100	BRTL	233.12	0.946717	880.3	30
	2PPLS	**233.35**	**0.946786**	**2473.0**	**13**
kroAB200	BRTL	835.37	0.8753578	1084.9	30
	2PPLS	**1076.08**	**0.945067**	**6736.5**	**20**
kroABC50	BRTL	*4092608*	*0.982856*	4888.2	**30**
	2PPLS(30)	3454695	0.97029	*10131.0*	**30**
	2PPLS	–	–	**40790+**	3600+

be interesting to observe the behavior of the algorithm on instances with other objectives, for instance ones which produce non-convex Pareto fronts (Li and Landa-Silva 2008).
- We use a fixed population size of 20 (unless the initial population is smaller, in which case it grows up to 20). Varying the size of the population through out the search might prove beneficial since this will prevent the algorithm from always converging to the same points which optimize hypervolume.
- The running population is randomly selected from the set of non-dominated solutions already found. This selection could be performed more intelligently by selecting solutions which are more likely to give better improvements (solutions with few neighbors for example).
- The punishment and reward mechanism is very simple and could be improved to take into account factors such as running time and percentage change in the quality of solutions. This "tuning" of the algorithm, which may need to be changed depending on the problem to solve, detracts from the relatively simple easy-to-use approach which hyperheuristics are supposed to be.
- The hypervolume metric, which drives the search, could be replaced or used in conjunction with other metrics.
- Our results show that BRTL performs comparatively better using an average starting population. We do not use the full potential of phase 1. The latter could be replaced by a simpler method which creates a number of solutions on the Pareto front using uniformly distributed weight vectors.
- Obviously, further investigations should try to include other low-level heuristics and find out how the size of the heuristic population can affect the resolution.

Hyperheuristics and assimilated heuristic-selection mechanisms have up to now been used for problems, such as timetabling, which are much more complicated than the TSP. Following the relatively poor performance of our approach for the TSP, and while we acknowledge that our method may not be better in another case study, it is our view that hyperheuristics are better suited to more difficult problems:

– Highly constrained problems are more likely to have low-level moves which will affect positively one constraint and others negatively. This bars the possibility of one heuristic performing very well on its own and thus a heuristic-selection mechanism is useful to choose the good move at the good moment.
– Problems such as timetabling cannot be readily be tackled with algorithms similar to Pareto local search (and in particular 2PPLS) which already offer very good results for simpler problems.

Despite what we have already stated, we believe that further investigation of hyperheuristics applied to the TSP needs to be done. One potential avenue would be using a GA-based hyperheuristic to evolve combinations of moves (as has been done "manually" by Babin et al. [2005]) to solve the single-objective TSP.

References

Aneja, Y.P., Nair, K.P.K.: Bicriteria Transportation Problem. Management Science, 25, pp. 73–78 (1979)

Angel, E., Bampis, E., Gourvès, L.: A dynasearch neighborhood for the bicriteria traveling salesman problem. In Gandibleux, X., Sevaux, M., Sörensen, K., T'kindt, V. (eds.) Metaheuristics for Multiobjective Optimisation. Lecture Notes in Economics and Mathematical Systems, 535, pp. 153–176. Springer, Berlin (2004)

Babin, G., Deneault, S., Laporte,G.: Improvements to the Or-opt Heuristic for the Symmetric Traveling Salesman Problem. In Cahier du GERAD G-2005-02, (2005)

Bentley, J.: Fast algorithms for the geometric traveling salesman problems. ORSA Journal on Computing 4, pp. 387–411 (1992)

Blum, C., Roli, A.: Metaheuristics in Combinatorial Optimization: Overview and Conceptual Comparison. ACM Computing Surveys, 35, pp. 268–308 (2003)

Effective Local and Guided Variable Neighbourhood Search Methods for the Asymmetric Travelling Salesman Problem. In Applications of Evolutionary Computing, LNCS 2037, pp. 203-212. Springer Berlin Heidelberg (2001)

Burke, E.K., Hart, E., Kendall, G., Newall, J., Ross, P., Schulenburg, S.: Hyper-heuristics: An Emerging Direction in Modern Search Technology. In Glover, F., Kochenberger, G.A. (eds.) Handbook of Metaheuristics, chapter 16, pp. 457–474. Kluwer Academic Publishers (2003)

Burke, E.K., Landa-Silva, J.D., Soubeiga, E.: Multi-objective Hyper-heuristic Approaches for Space Allocation and Timetabling. In Ibaraki, T., Nonobe, K., Yaguira, M. (eds.) Meta-heuristics: Progress as Real Problem Solvers, pp. 129–158. Springer (2005)

Chakhlevitch, K., Cowling, P.: Hyperheuristics: Recent Developments. In Cotta, C., Sevaux, M., Sörensen, K. (eds.) Adaptative and Multilevel Metaheuristics, pp. 3–29. Springer-Verlag Berlin Heidelberg (2008)

Cowling, P., Chakhlevitch, K.: Hyperheuristics for Managing a Large Collection of Low Level Heuristics to Schedule Personnel. In Proceedings of the 2003 IEEE Congress on Evolutionary Computation (CEC 2003), pp. 1214–1221. IEEE Press (2003)

Cowling, P., Kendall, G., Soubeiga, E.: A Hyperheuristic Approach to Scheduling a Sales Summit. In Burke, E., Erben, W. (eds.) PATAT 2000 LNCS, 2079, pp. 176–190. springer, Heidelberg (2001)

Ehrgott, M., Gandibleux X.: A survey and annotated bibliography of multiobjective combinatorial optimization. In OR Sprectrum 22, pp. 425-460 (2002)

Fang, H.L, Ross, P., Corne, D.: A Promising Hybrid GA/Heuristic Approach for Open Shop Scheduling Problems. In Proceedings of the 11th European Conference on Artificial Intelligence, pp. 590-594. John Wiley and Sons (1994)

Farhang-Mehr, A., Azarm, S.: An Information-Theoretic Entropy Metric for Assessing Multi-Objective Optimization Solution Set Quality. Journal of Mechanical Design, 125, pp. 655–663 (2003)

Fonseca, C.M., Paquete, L., López-Ibáñez, M.: An improved dimension-sweep algorithm for the hypervolume indicator. In Proceedings of the IEEE Congress on Evolutionary Computation, pp. 1157–1163. IEEE Press (2006)

Gandibleux, X., Morita, H., Katoh, N.: Use of a genetic heritage for solving the assignment problem with two objectives. In Evolutionary Multi-Criterion Optimization, LNCS 2632, pp. 43–57, Springer Berlin Heidelberg (2003)

Glover, F., Kochenberger, G.A. (eds.): Handbook of Metaheuristics. Kluwer Academic Publishers (2003)

Gutin, G., Punnen, A.P. (eds.): The Traveling Salesman Problem and Its Variations. Kluwer Academic Publishers (2002)

Jaszkiewicz, A.: On the performance of multiple-objective genetic local search on the 0/1 knapsack problem – a comparative experiment. Technical Report RA-002/2000, Institute of Computing Science, Poznan University of Technology, Poznań, Poland (2000)

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

Jaszkiewicz, A., Zielniewicz, P.: Pareto memetic algorithm with path relinking for bi-objective traveling salesperson problem. European Journal of Operational Research 193, pp. 885–890 (2009)

Johnson, D.S., McGeoch L.A.: The Traveling Salesman Problem: A Case Study in Local Optimization. In Aarts, E.H.L., Lenstra, J.K. (eds.) Local Search in Combinatorial Optimization. John Wiley and Sons, London, pp. 215–310 (1997)

Kaelbling, L.P., Littman, M.L., Moore, A.W.: Reinforcement Learning: A Survey Journal of Artificial Intelligence Research 4, pp. 237–285 (1996)

Kendall, G., Modh Hussin, N.: A Tabu Search Hyper-heuristic Approach to the Examination Timetabling Problem at the MARA University of Technology. In Burke, E.K., Trick, M. (eds.) PATAT 2004, LNCS, 3616, pp.199–217. Springer, Heidelberg (2005)

Li, H., Landa-Silva, J.D.: Evolutionary Multi-objective Simulated Annealing with Adaptive and Competitive Search Direction. In Proceedings of the 2008 IEEE Congress on Evolutionary Computation (CEC 2008), pp. 3310–3317. IEEE Press (2008)

Lin, S.: Computer Solutions of the Traveling Salesman Problem. Bell System Technical Journal, 44, pp. 2245-2269 (1965).

Lin, S., Kernighan, B.W.: An effective heuristic algorithm for the traveling salesman problem. Operations Research, 21, pp. 972–989 (1973)

Lust, T., Teghem, J.: Two-phase Pareto local search for the biobjective Traveling salesman problem. J. Heuristics. Springer Netherlands (2009)

Nareyek, A.: Choosing search heuristics by non-stationary reinforcement learning. In Resende, M., de Sousa, J. (eds.) Metaheuristics: Computer decison-making, pp. 523–544. Kluwer Academic Publishers (2003)

Okabe, T., Jin, Y., Sendhoff, B.: A Critical Survey of Performance Indices for Multi-Objective Optimisation. In Proceedings of IEEE Congress on Evolutionary Computation (CEC-2003), pp. 878–885. IEEE Press (2003)

Or, I.: Traveling Salesman-Type Combinatorial Problems and Their Relation to the Logistics of Regional Blood Banking. Ph.D. Thesis, Department of Industrial Engineering and Management Sciences, Northwestern University, Evanston, IL (1976)

Paquete, L., Chiarandini, M., Stützle, Th.: Pareto local optimum sets in the biobjective traveling salesman problem: an experimental study. In Gandibleux, X., Sevaux, M., Sörensen, K., T'kindt, V. (eds.) Metaheuristics for Multiobjective Optimisation. Lecture Notes in Economics and Mathematical Systems, 535, pp. 177–199. Springer, Berlin (2004)

Paquete, L., Stützle, Th.: A Two-Phase Local Search for the Biobjective Traveling Salesman Problem. (2003)

Paquete, L., Stützle, Th.: Design andanalysis of stochastic local search for the multiobjective traveling salesman problem. Computers & OperationsResearch 36, pp. 2619–2631 (2009)

Przybylski, A., Gandibleux, X., Ehrgott, M.: A Two Phase Method for Multi-objective Integer Programming and its Application to the Assignment Problem with Three Objectives. To appear in INFORMS Journal on Computing (2009)

Soubeiga, E.: Development and Application of Hyperheuristics to Personnel Scheduling. PhD Thesis, University of Nottingham, UK (2003)

Stattenberger, G., Dankesreiter, M., Baumgartner, F., Schneider, J.J.: On the Neighborhood Structure of the Traveling Salesman Problem Generated by Local Search Moves. J. Stats. Phys. 129, pp. 623–648 (2007)

Zitzler, E., Deb, K., Thiele, L.: Comparison of Multiobjective Evolutionary Algorithms: Empirical Results. In Evolutionary Computation, 8(2), pp. 173–195 (2000)

Zitzler, E., Thiele, L.: Multiobjective evolutionary algorithms: a comparative case study and the strength Pareto approach. In IEEE Transactions on Evolutionary Computation, 3, pp. 257–271 (1999)

Zitzler, E., Thiele, L., Laumanns, M., Fonseca, C.M., Grunert da Fonseca, V.: Performance Asessment of Multiobjective Optimizers: An Analysis and Review. In IEEE Transactions on Evolutionary Computing, 7, pp. 117–132 (2003)