DEPARTMENT OF MATHEMATICS,
STATISTICS AND ACTUARIAL SCIENCES

An efficient GRASP+VND metaheuristic for the traveling repairman problem

Amir Salehipour, Kenneth Sörensen, Peter Goos & Olli Bräysy

UNIVERSITY OF ANTWERP
Faculty of Applied Economics

EARNED EXCELLENCE
AACSB ACCREDITED
EST. 1916
THE BEST BUSINESS SCHOOLS IN THE WORLD

Stadscampus
Prinsstraat 13, B.213
BE-2000 Antwerpen
Tel. +32 (0)3 220 40 32
Fax +32 (0)3 220 47 99
http://www.ua.ac.be/tew

FACULTY OF APPLIED ECONOMICS

DEPARTMENT OF MATHEMATICS,
STATISTICS AND ACTUARIAL SCIENCES

An efficient GRASP+VND metaheuristic for the traveling repairman problem

Amir Salehipour, Kenneth Sörensen, Peter Goos & Olli Bräysy

RESEARCH PAPER 2008-008
MAY 2008

University of Antwerp, City Campus, Prinsstraat 13, B-2000 Antwerp, Belgium
Research Administration – room B.213
phone: (32) 3 220 40 32
fax: (32) 3 220 47 99
e-mail: joeri.nys@ua.ac.be

The papers can be also found at our website:
www.ua.ac.be/tew
(research > working papers)

D/2008/1169/008

An efficient GRASP+VND metaheuristic for the traveling repairman problem

Amir Salehipour[1], Kenneth Sörensen[2], Peter Goos[1], and Olli Bräysy[3]

[1]Department of Mathematics, Statistics and Actuarial Sciences, Faculty of Applied Economics, University of Antwerp, Prinsstraat 13, 2000 Antwerp, Belgium, amir.salehipour@ua.ac.be, peter.goos@ua.ac.be
[2]Centre for Industrial Management, KULeuven, Celestijnenlaan 300a, 3001 Leuven, Belgium, kenneth.sorensen@cib.kuleuven.be
[3]Agora Innoroad Laboratory, Agora Center, University of Jyväskylä, P.O.Box 35, FI-40014, Finland, olli.braysy@jyu.fi

January 2008

The traveling repairman problem is a customer-centric routing problem, in which the total waiting time of the customers is minimized, rather than the total travel time of a vehicle. To date, research on this problem has focused on exact algorithms and approximation methods. To the best of our knowledge, this paper presents one of the first metaheuristic approaches for the traveling repairman problem.

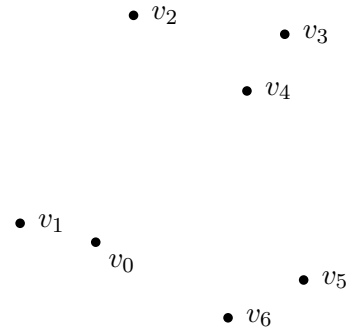
Key words: Traveling Repairman Problem, Minimum Latency Problem, Variable Neighborhood Descent, GRASP

1 Introduction

The traveling repairman problem (TRP), also called the minimum latency problem or the traveling deliveryman problem, is defined on a weighted graph $G = (V, E, w)$. The vertex set $V = \{v_0, \dots v_n\}$ represents a set of locations, that have to be visited, starting from v_0. With each edge $(v_i, v_j) \in E$, a weight $w(v_i, v_j)$ is associated, that represents the distance between v_i and v_j. Sometimes, an additional time t_k is associated with each vertex k, to represent the time spent at this vertex. In the symmetric TRP, travel times between customers are the same in both directions, i.e. $w(v_i, v_j) = w(v_j, v_i)$.

The objective of the TRP is to find an open Hamiltonian circuit (or Hamiltonian path), starting from v_0, that minimizes the total waiting time. The waiting time of a

vertex, also called its latency, is defined as the time it takes to reach it. The TRP can be seen as a "customer-centric" routing problem, because the objective function minimizes the total waiting time of all customers, rather than the travel time of the vehicle the customers is visited with. Besides the obvious applications in customer-centric routing (e.g. a doctor visiting patients, or a repairman visiting machines that require servicing), the TRP has applications in e.g. disk head scheduling (Blum et al., 1994). An example traveling repairman problem, including a (symmetric) distance matrix, can be found in Fig. 1.



	v_1	v_2	v_3	v_4	v_5	v_6
v_0	2.1	6.1	7.4	5.7	5.6	4.0
v_1		6.3	8.6	6.9	7.6	6.0
v_2			4.0	3.6	8.3	8.4
v_3				1.8	6.5	7.6
v_4					5.2	6.0
v_5						2.2

Fig. 1 – Example traveling repairman problem and (symmetric) distance matrix

Although the TRP may seem very similar to the traveling salesman problem (TSP), its formulation is more involved (see e.g. Sarubbi and Luna (2007) for a flow-based integer programming formulation). On a metric space, the TRP is NP-hard (Blum et al., 1994) like the TSP but surprisingly, the TRP is much harder to solve or approximate. Polynomial time algorithms exist in a number of special cases, such as paths (Afrati et al., 1986; García et al., 2002) or trees of diameter 3 (Blum et al., 1994). A lot of research has gone into developing approximations for the TRP, and the current best approximation ratio is 3.59 (Chaudhuri et al., 2003). In contrast, the best-known approximation ratio for the metric TSP is 1.5, due to Christofides (1976).

Several exact algorithms for the TRP exist, such as the dynamic programming algorithm of Wu (2000) or the much more efficient branch and bound algorithm of Wu et al. (2004). Due to the NP-hardness of the TRP, however, these approaches are unable to solve medium- and large-scale instances. The fastest method of Wu et al. (2004) requires 100 seconds to solve a problem with approximately 25 vertices.

Research on the TRP has focused almost exclusively on either approximation algorithms or exact methods. To the best of our knowledge, no method therefore exists to efficiently solve large-scale instances. In this paper, we develop the first metaheuristic approach for this problem and test it on a large set of generated instances. Our GRASP+VND approach is a multi-start method in which each iteration consists of two phases: a greedy randomized construction phase and a variable neighborhood descent improvement phase.

GRASP (*Greedy Randomized Adaptive Search Procedure*) was introduced by Feo and

Resende (1989, 1995). The basic idea of GRASP is to allow a controlled amount of randomness to overcome the myopic behaviour of a purely greedy heuristic. This method has been applied to a large number of combinatorial optimization problems, including routing problems such as the vehicle routing problem with time windows (Kontoravdis and Bard, 1995). Although initial implementations of GRASP combined it with simple local search strategies, more recent work has begun to combine it with more advanced ones.

Among those more advanced local search schemes is *variable neighborhood search* (VNS) (Mladenović and Hansen, 1997; Hansen and Mladenović, 1997, 1999), a recent metaheuristic for combinatorial optimization. VNS is based on the principle of systematically exploring several different neighborhoods, combined with a perturbation move (called *shaking* in the VNS literature) to escape from local optima. Despite it being a relatively recent development, variable-neighborhood search has been successfully applied to a wide variety of well-known and lesser-known optimization problems (Hansen and Mladenović, 2001a,b). Several researchers have applied some form of variable neighborhood search to (more or less academic) vehicle routing problems. Two examples are Cordone and Wolfer-Calvo (2001), in which a variable neighborhood search algorithm for the vehicle routing problem with time windows is developed, and Crispim and Brandao (2001), who present a VNS approach to the vehicle routing problem with backhauls.

Variable neighborhood *descent* (VND) is essentially a simple variant of VNS, in which the shaking phase is omitted. Therefore, contrary to VNS, VND is usually completely deterministic. The combination of GRASP and VND is not new: Ribeiro and Vianna (2003), for example, applied it to the phylogeny problem.

It is important however to note that the concept of *multiple* neighborhood search (i.e. using more than one neighborhood type) is broader than that of *variable* neighborhood search and that a large number of non-VNS approaches use different neighborhood types. In the list of best-performing approaches for most vehicle routing problems, the idea of multiple neighborhood search is overwhelmingly present. In their survey article on metaheuristics for the vehicle routing with time windows, Bräysy and Gendreau (2005) find that a large majority of tabu search approaches and genetic algorithms use more than one type of neighborhood.

The rest of this paper is organized as follows. In section 2, we discuss our GRASP+VND for the TRP. To evaluate the performance of this approach, we discuss an upper and a lower bound in section 3. A set of benchmark instances is generated in section 4. Section 5 contains details on the experiments that were performed to test our algorithm. Finally, conclusions and future research directions are given in section 6.

2 GRASP+VND for the TRP

As mentioned in the introduction, our method for the TRP combines a GRASP construction phase with a VND improvement phase. Both phases are repeated a number of times, and the best solution found is reported. An outline of the algorithm is shown in Algorithm 1.

Algorithm 1: GRASP+VND for the TRP outline

```
for number of iterations allowed do
    find initial solution using GRASP;
    improve solution using VND;
report best solution found;
```

2.1 Solution representation and objective function calculation

A solution in our algorithm is represented as a permutation of the vertices from 1 to n. From this representation, the objective function value can be easily calculated. Assume a solution $x = x_1, x_2, \ldots, x_n$ with x_1 being the first vertex in the solution, x_2 the second, and so on. The total latency for this solution can be calculated as

$$f(x) = \sum_{i=1}^{n} (n - i + 1) w(x_{i-1}, x_i), \tag{1}$$

keeping in mind that $x_0 \equiv v_0$. This calculation is equivalent to the more intuitive calculation in which the latencies of the vertices in the solution are summed:

$$f(x) = \sum_{i=1}^{n} l(x_i) \text{ where } l(x_i) = \sum_{j=1}^{j \leqslant i} w(x_{j-1}, x_j). \tag{2}$$

Eq. (2) offers the advantage of making the calculations easier when e.g. vertices are added or deleted.

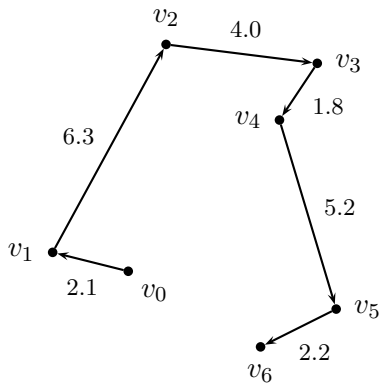


Fig. 2 – Example traveling repairman problem solution

In Fig. 2, we show an example solution for the TRP problem in Fig. 1. The objective function of this solution is calculated as

$$f(x) = 6(2.1) + 5(6.3) + 4(4.0) + 3(1.8) + 2(5.2) + 1(2.2) = 78.1.$$

Many of the heuristics developed in the following sections require finding the closest, second-closest, etc. vertex to any given vertex. We therefore pre-process the distance

data and maintain in memory a *proximity matrix* that contains one row per vertex. The $i+1$-th row of the matrix contains all vertices (except for vertex v_i) in order of proximity to vertex v_i. For the example in Fig. 1, the proximity matrix is shown in Fig. 3.

	1st	2nd	3rd	4rd	5th	6th
v_0	v_1	v_6	v_5	v_4	v_2	v_3
v_1	v_0	v_6	v_2	v_4	v_5	v_3
v_2	v_4	v_3	v_0	v_1	v_5	v_6
v_3	v_4	v_2	v_5	v_0	v_6	v_1
v_4	v_3	v_2	v_5	v_0	v_6	v_1
v_5	v_6	v_4	v_0	v_3	v_1	v_2
v_6	v_5	v_0	v_4	v_1	v_3	v_2

Fig. 3 – Proximity matrix for the example in Fig. 1

2.2 GRASP

The motivation for using GRASP as a procedure for the TRP is the observation that the initial choices of vertices to visit, i.e. the first few vertices in the tour, are more important than later ones. This immediately follows from Eq. (1), from which it is clear that the i-th distance in the solution is multiplied with a factor $n - i + 1$. We therefore expect a greedy algorithm to perform well, as it attempts to add the shortest possible distances early on and leaves the longer ones for later. A completely greedy algorithm on the other hand, can be expected to miss some interesting opportunities.

Perhaps the most distinguishing characteristic of GRASP is the way in which it combines greediness with randomness in the construction phase. To this end, a *restricted candidate list* (RCL) is built by selecting a subset of all elements in a greedy fashion. Assuming a minimization problem, the RCL contains the elements whose incorporation into the partially built solution would yield the smallest increase in objective function value. From the RCL, an element is then selected at random, after which the RCL is updated to reflect the fact that a new element was added to the solution and is no longer available for selection. Selection of an element and update of the RCL are repeated until a completed solution has been built. From this solution, a local search phase starts until a local optimum is found. The size of the RCL, α, is a parameter of the GRASP algorithm that controls the balance between greediness and randomness. If α is small, the search is relatively greedy. If α is large, it is relatively random. In the extreme cases, setting $\alpha = 1$ leads to a completely deterministic greedy search, whereas setting $\alpha = n$ entails a completely random search.

The RCL concept can be easily translated to the TRP problem, and efficiently implemented using the proximity matrix (see Fig. 3). Starting from vertex v_0, the RCL is filled with the α closest vertices. From this RCL, a random vertex is chosen, say v_r. Now, the RCL is filled with the α vertices closest to v_r, omitting any elements that have already been selected. If less than α vertices remain, then the size of the RCL is

decreased. A pseudo-code version of the GRASP construction phase of our algorithm for the TRP is in Algorithm 2.

Algorithm 2: GRASP for the TRP

$U \leftarrow \{v_1, v_2, \ldots, v_n\}$;
$v_c \leftarrow v_0$;
repeat
  create RCL with α vertices $v_i \in U$ closest to v_c;
  select random vertex $v_r \in$ RCL;
  $U \leftarrow U \backslash \{v_r\}$;
  $v_c \leftarrow v_r$;
until $U = \emptyset$;

2.3 Variable neighborhood descent

As its name suggests, this metaheuristic systematically explores different *neighborhood structures*. The main idea underlying VNS and also VND is that a local optimum relative to a certain neighborhood structure is not necessarily a local optimum relative to another neighborhood structure. For this reason, escaping from a local optimum can be done by changing the neighborhood structure.

Although this is not required, many implementations of VNS use a sequence of *nested* neighborhoods, $\mathcal{N}_1$ to $\mathcal{N}_{k_{\max}}$, in which each neighborhood in the sequence is a superset of its predecessor, i.e. $\mathcal{N}_k \subset \mathcal{N}_{k+1}$. The neighborhoods used in our VND do not possess this property.

Pseudo-code for a basic version of VND is given in algorithm 3.

Algorithm 3: Basic variable neighborhood descent

Input: initial solution x
initialise: $k \leftarrow 1$;
repeat
  local search: $x' \leftarrow \arg\min \mathcal{N}_k(x)$;
  if x' *is better than* x **then**
    $x \leftarrow x'$ and $k \leftarrow 1$ (centre the search around x' and search again with the current neighborhood);
  else
    $k \leftarrow k + 1$ (switch to another neighborhood);
until $k = k_{max}$;

Usually, variable-neighborhood search algorithms examine neighborhoods in a certain order, from small to large. The reason is that small neighborhoods contain fewer solutions and can hence be searched in less time than large neighborhoods. Larger neighborhoods are only examined when all smaller neighborhoods have been depleted,

i.e. when the current solution is a local optimum of all smaller neighborhoods. The variable neighborhood descent stops when the solution is a local optimum of all neighborhoods. Another option is to use the neighborhoods in the order of their effectiveness with respect to the problem at hand, which can be established e.g. by means of a small pilot study.

2.3.1 Neighborhoods

Our search uses five neighborhood structures: swap, swap-adjacent, 2-opt, Or-opt, and remove–insert.

The *swap* heuristic attempts to swap the positions of each pair of vertices in the solution (see Fig. 4). Searching the entire neighborhood of a solution with this heuristic requires quadratic time in the problem size ($O(n^2)$). Using Eq. (1), examining the effect of a swap of vertices x_i and x_j in solution x can be easily done by substracting the weight of the removed edges and adding that of the added ones.

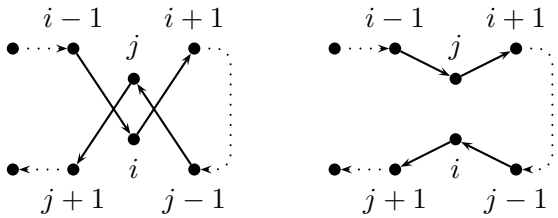


Fig. 4 – Swap

The *swap-adjacent* heuristic, as its name suggests, attempts to swap each pair of adjacent vertices in the solution x. This heuristic is a subset of the swap heuristic and has a linear complexity ($O(n)$).

The *2-opt* heuristic (see Fig. 5) removes each pair of edges from the solution and reconnects the vertices. Like the swap heuristic, this heuristic has quadratic complexity. Updating the objective function is more complicated for 2-opt, since the middle part of the solution (that was "cut out") is reversed. This heuristic requires $O(n^2)$ time to examine the entire neighborhood of a solution.

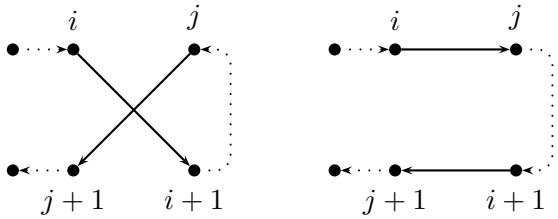


Fig. 5 – 2-opt

The *Or-opt* heuristic (see Fig. 6) removes each triplet of edges from the solution and reconnects the solution in such a way that the orientation of the solution parts is preserved. The time complexity of this heuristic is cubic ($O(n^3)$).

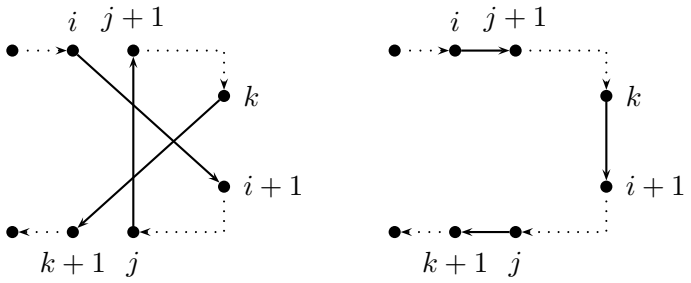


Fig. 6 – Or-opt

The *remove–insert* heuristic examines each vertex x_i in the solution and places the vertex furthest away from x_i at the end of the solution. This heuristic has linear complexity like the swap-adjacent heuristic.

2.3.2 Neighborhood reduction

Several attempts are made to decrease the size of the neighborhoods to examine during the local search phase. All neighborhood reduction schemes are carefully tested in section 5.

For the swap, 2-opt and Or-opt moves the neighborhood size is reduced by fixing a random vertex and then attempting only the moves that involve this vertex. We call this strategy "Fix" and compare it to the strategy in which the entire neighborhood is searched without first fixing a random vertex, which we call "Nofix". This neighborhood reduction scheme decreases the size of the neighborhood by a factor $O(n)$.

A second neighborhood reduction scheme is based on the observation that in a reasonably good solution, most improving moves will involve vertices that appear in relative proximity to each other in this solution. For example, in a good solution it is unlikely that a swap will yield a better solution if it exchanges a vertex that appears close to the depot with one that appears far from the depot. We therefore introduce a *proximity factor* β to determine the maximum distance between vertices that may be involved in a move. The factor β has a slightly different influence, depending on the move. For a swap move, the second vertex may be only β positions further in the solution than the first one. For a 2-opt move, the *head* of the second chosen vertex may only be β positions further in the solution than the head of the first chosen vertex. For an Or-opt move, the heads of both the second and third chosen vertices may be only β positions away from the head of the first vertex. For the swap-adjacent and remove–insert heuristics, which have linear size, this strategy is not used as it would reduce their size to 1.

3 Bounds for the TRP

As our procedure is the first metaheuristic for the TRP, it is impossible to compare the performance of our approach to that of other published results. We therefore compare the outcome of our algorithm to an upper and a lower bound, discussed in this section.

3.1 Upper bound

Like Simchi-Levi and Berman (1991) and Bianco et al. (1993), we use the nearest neighbor heuristic to derive an upper bound for the TRP. The nearest neighborhood heuristic belongs to the family of greedy algorithms and constructs an initial solution by starting from the depot (v_0) and repeatedly adding the closest vertex to the last-added vertex until all vertices have been added. It is a special case of our GRASP procedure, in which the RCL only contains the closest vertex not yet added to the solution, i.e. in which $\alpha = 1$. As an example, application of the nearest neighbor heuristic to the example in Fig. 1 yields the solution $v_1v_6v_5v_4v_3v_2$.

3.2 Lower bound

Our lower bound for the TRP is a variant of the minimum spanning tree (MST) lower bound for the TSP. It is computed by sorting the edges of the MST of the graph in order of increasing weight and multiplying each edge with a factor like the edges of the TRP solution in Eq. (1). The smallest edge is multiplied with n, the second-smallest with $n-1$, etc. The largest edge is multiplied with a factor 1.

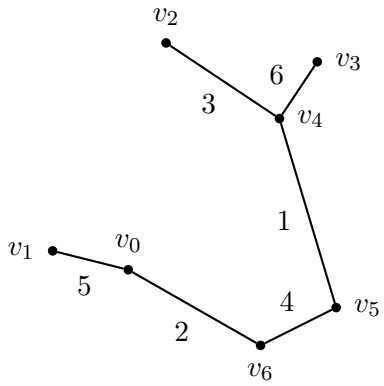


Fig. 7 – Lower bound for the problem in Fig. 1

A lower bound for the example in Fig. 1 is shown in Fig. 7. Besides each edge, the multiplication factor is shown. The value of this lower bound is

$$\text{LB} = 6(1.8) + 5(2.1) + 4(2.2) + 3(3.6) + 2(4.0) + 1(5.2) = 54.1.$$

Corollary 1. *Let $t = t_1, t_2, \ldots, t_n, t_i \in E$ be the edges of the minimum spanning tree of the graph $G = (V, E, w)$, sorted in increasing order of weight, i.e. $t_i \leqslant t_j$ iff $i < j$. A*

lower bound for the TRP on G is given by

$$LB = \sum_{i=1}^{n} (n - i + 1) w(t_i), \tag{3}$$

where $w(t_i)$ is the weight of edge t_i.

Proof. Assume that the optimal traveling repairman solution of $G = (V, E, w)$ is given by $r = r_1, r_2, \ldots, r_n, r_i \in E$, with the edges in order of appearance in the solution. The objective function value of r is $f(r) = \sum_{i=1}^{n} (n - i + 1) w(r_i)$.

Since t is a minimum spanning tree, the edges $t_1, t_2, \ldots, t_i$ are the smallest edges of the graph that do not form a cycle. It holds that the sum of the weights of the i smallest edges that do not form a cycle, is smaller than the sum of the weights of any other i edges that do not form a cycle. Since the first i edges in the traveling repairman solution also do not form a cycle, it follows that

$$\sum_{j=1}^{i} w(t_j) \leqslant \sum_{j=1}^{i} w(r_j).$$

Summing over i, we get that

$$\sum_{i=1}^{n} \sum_{j=1}^{i} w(t_j) \leqslant \sum_{i=1}^{n} \sum_{j=1}^{i} w(r_j),$$

which simplifies to

$$\sum_{i=1}^{n} (n - i + 1) w(t_i) \leqslant \sum_{i=1}^{n} (n - i + 1) w(r_i).$$

□

4 Benchmark instance set generation

To the best of our knowledge, no standard set of medium- and large-scale benchmark instances for the TRP exists. Therefore, we have generated an extensive set of 140 symmetric instances in seven different problem sizes ranging from 10 to 1000 vertices (10, 20, 50, 100, 200, 500, 1000). For each problem size, we have generated 20 random instances. For each instance, vertex coordinates have been generated from a uniform distribution between 0 and 100 (or between 0 and 500 for the instances of size 500 and 1000). All distances are Euclidean, rounded down to the nearest integer. The instances are available from the authors upon request.

Table 1 – Random instances for the TRP

File Name	Size	[min,max]
TRP-S10-Rx	10	(0,100)
TRP-S20-Rx	20	(0,100)
TRP-S50-Rx	50	(0,100)
TRP-S100-Rx	100	(0,100)
TRP-S200-Rx	200	(0,100)
TRP-S500-Rx	500	(0,500)
TRP-S1000-Rx	1000	(0, 500)

5 Experiments

Several parameters need to be set for our algorithm. Concerning the construction phase, a pilot study clearly showed that the greediness of the GRASP method needs to be large, i.e. values of $\alpha > 2$ only very rarely yielded better results than those found for the same problem with $\alpha \leqslant 2$. As mentioned, the importance of adding the smallest edges early on in a construction procedure is not surprising given their importance in the objective function. We therefore test our procedure with $\alpha = 2$.

The first neighborhood reduction scheme is tested by running the algorithm with both implementations of the neighborhoods. Runs in which the search starts by selecting a fixed random vertex, are labeled "Fix". Runs in which the search attempts all possible moves are labeled "Nofix".

The implementation of the second neighborhood reduction scheme depends on the size of the problem. For problems of size $n < 500$, a "Full" strategy is tested with $\beta = n$ against a "Half" strategy with $\beta = n/2$. For problems with 500 and 1000 vertices, β equals 250 in the Full strategy for swap, 2-opt and Or-opt. In the Half strategy, β equals 300, 200 and 100 for swap, 2-opt and Or-opt respectively. The settings of β were obtained using a small pilot study. For problems of size 10, the Half-strategy is ignored.

We also compare a *multi-start* version of our algorithm, in which 10 solutions are generated with the GRASP procedure with a single-start procedure in which only one solution is generated.

The two neighborhood reduction schemes are combined with the single- and multi-start setting, giving rise to eight different settings of our algorithm. These eight settings are shown in Table 2.

All the computations reported in this section have been carried out on a Personal Computer with Pentium 4, 2.4 GHz processor and 512 MB RAM. The algorithm was coded in C++.

The results of our algorithm on the different data sets are reported in Table 3. In these tables, the column UB% shows the improvement of the objective function over the upper bound obtained with the nearest neighbor heuristic. The column LB% shows the gap to the lower bound, obtained by the spanning tree with edge multiplicator. In column T, the computation time in seconds is shown.

Table 2 – Experiment settings

Restarts	Neighborhood implementation	Proximity rule	Experiment number
Single-start	Nofix	Full	1
		Half	2
	Fix	Full	3
		Half	4
Multi-start	Nofix	Full	5
		Half	6
	Fix	Full	7
		Half	8

From Table 3 we can draw some conclusions about the working of our algorithm. Unsurprisingly, the multi-start version of our algorithm (algorithm settings 5 to 8) requires about 10 times the computation time of the single-start version (settings 1 to 4). However, the quality improvement obtained by this method is relatively small.

Both neighborhood reduction strategies seem to work well. The neighborhood implementation with fixed random vertex (Fix, settings 1, 2, 5, and 6) uses significantly less computing time, combined with a rather small loss of quality. The Half neighborhood reduction strategy (settings 2, 4, 6, and 8) yields a much faster solution method, at the cost of some loss in objective function quality. Both neighborhood reduction strategies prove useful for the largest instances (500 and 1000 vertices), for which the setting with full neighborhood search (Full and NoFix) proved too time consuming.

6 Conclusions and future research

In this paper, we have presented the first metaheuristic for the traveling repairman problem. This metaheuristic consists of a GRASP construction phase and a variable neighborhood descent improvement phase. Several neighborhood reduction schemes were also introduced to speed up the search. To test the approach, we generated a set of medium-sized and large instances and discussed an upper and a lower bound. Experiments show that our approach is able to solve the generated instances in a reasonable amount of time.

In the future, we intend to extend our algorithm by including more neighborhoods and carefully studying the effectiveness of each neighborhood on the TRP. We also intend to apply our algorithm to other customer-centric routing problems, involving more than one vehicle. Increasing the efficiency of our algorithm even more, to allow even larger problems to be solved, is another future research point. Finally, the integration of exact methods for the TRP might lead to even better performing algorithms and is something that we plan to investigate.

Table 3 – Results of the GRASP+VND

	Single-start					
	1 (Nofix-Full)			2 (Nofix-Half)		
	UB%	LB%	T	UB%	LB%	T
10	2.14	33.48	0.00	-	-	-
20	8.19	42.43	0.00	6.89	44.75	0.00
50	6.85	50.07	0.27	6.70	50.21	0.03
100	10.24	45.53	9.73	9.99	45.97	1.21
200	10.69	43.37	356.73	10.20	44.18	39.81
500	10.70	46.49	7192.94	10.49	46.84	635.96
1000	-	-	-	11.32	46.40	3378.28
	Single-start					
	3 (Fix-Full)			4 (Fix-Half)		
	UB%	LB%	T	UB%	LB%	T
10	1.96	33.71	0.00	-	-	-
20	7.51	43.45	0.00	6.51	45.19	0.00
50	5.69	51.95	0.01	5.24	52.64	0.00
100	8.55	48.33	0.32	8.56	48.30	0.07
200	8.61	46.77	6.30	8.54	46.88	1.25
500	9.19	48.97	134.83	9.18	49.00	18.86
1000	9.31	49.72	837.26	9.22	49.88	112.99
	Multi-start					
	5 (Nofix-Full)			6 (Nofix-Half)		
	UB%	LB%	T	UB%	LB%	T
10	2.44	33.04	0.00	-	-	-
20	9.86	39.63	0.04	9.80	39.80	0.00
50	9.67	45.41	3.21	9.48	45.71	0.43
100	11.56	43.40	101.27	11.21	43.93	12.55
200	11.33	42.32	3741.05	11.44	42.14	444.43
500	8.11	50.73	91470.78	11.47	45.22	6335.79
1000	-	-	-	8.18	51.57	9114.75
	Multi-start					
	7 (Fix-Full)			8 (Fix-Half)		
	UB%	LB%	T	UB%	LB%	T
10	2.44	33.04	0.00	-	-	-
20	9.41	40.34	0.01	9.70	39.82	0.00
50	8.29	47.59	0.19	7.83	48.40	0.04
100	8.43	48.43	3.54	8.80	47.82	0.80
200	7.75	48.07	78.25	7.76	48.05	16.56
500	8.11	50.73	1534.90	8.07	50.81	239.44
1000	-	-	-	8.09	51.73	1431.16

References

F. Afrati, S. Cosmadakis, C. H. Papadimitriou, G. Papageorgiou, and N. Papakostantinou. The complexity of the traveling repairman problem. *Theoretical Informatics and Applications*, 20:79–87, 1986.

L. Bianco, A. Mingozzi, and S. Ricciardelli. The travelling salesman problem with cumulative costs. *Networks*, 23(2):81–91, 1993.

A. Blum, P. Chalasani, D. Coppersmith, B. Pulleyblank, P. Raghavan, and M. Sudan. The minimum latency problem. In *Proceedings of the twenty-sixth annual symposium on Theory of Computing (STOC)*, pages 163–171, 1994.

O. Bräysy and M. Gendreau. Vehicle routing problem with time windows, part II: Metaheuristics. *Transportation Science*, 39:119–139, 2005.

K. Chaudhuri, B. Godfrey, S. Rao, and K. Talwar. Paths, trees, and minimum latency tours. In *Proceedings of 44th Symposium on Foundations of Computer Science (FOCS)*, pages 36–45, 2003.

N. Christofides. Worst-case analysis of a new heuristic for the travelling salesman problem. Technical Report 388, Graduate School of Industrial Administration, Carnegie-Mellon University, Pittsburgh, PA, 1976.

R. Cordone and R. Wolfer-Calvo. A heuristic for the vehicle routing problem with time windows. *Journal of Heuristics*, 7:107–129, 2001.

J. Crispim and J. Brandao. Reactive tabu search and variable neighborhood descent applied to the vehicle routing problem with backhauls. In *MIC 2001 – Proceedings of the Metaheuristics International Conference*, pages 631–636, Porto, 2001.

T.A. Feo and M.G.C. Resende. A probabilistic heuristic for a computationally difficult set covering problem. *Operations Research Letters*, 8:67–71, 1989.

T.A. Feo and M.G.C. Resende. Greedy randomized adaptive search procedures. *Journal of Global Optimization*, 6:109–133, 1995.

A. García, P. Jodrá, and J. Tejel. A note on the traveling repairman problem. *Networks*, 40(1):27–31, 2002.

P. Hansen and N. Mladenović. Industrial applications of the variable neighbourhood search metaheuristic. In *Decisions and Control in Management Science*, pages 261–274, Boston, 2001a. Kluwer.

P. Hansen and N. Mladenović. Variable neighbourhood search: Principles and applications. *European Journal of Operational Research*, 130:449–467, 2001b.

P. Hansen and N. Mladenović. Variable neighborhood search for the p-median. *Location Science*, 5:207–226, 1997.

P. Hansen and N. Mladenović. An introduction to variable neighborhood search. In S. Voss, S. Martello, I. Osman, and C. Roucairol, editors, *Metaheuristics: Advances and Trends in Local Search Paradigms for Optimization*, pages 433–458, Boston, 1999. Kluwer.

G.A. Kontoravdis and J.F. Bard. A GRASP for the vehicle routing problem with time windows. *INFORMS Journal on Computing*, 7:10–23, 1995.

N. Mladenović and P. Hansen. Variable neighbourhood decomposition search. *Computers and Operations Research*, 24:1097–1100, 1997.

C.C. Ribeiro and D.S. Vianna. A GRASP/VND heuristic for the phylogeny problem using a new neighborhood structure. Technical report, Department of Computer Science, Catholic U. of Rio de Janeiro, Rio de Janeiro, Brazil, 2003.

J. Sarubbi and H. Luna. A new flow formulation for the minimum latency problem. In *International Network Optimization Conference (INOC)*, Spa, Belgium, 2007.

D. Simchi-Levi and O. Berman. Minimizing the total flow time of n jobs on a network. *IIE Transactions*, 23(3):236–244, 1991.

B.Y. Wu. Polynomial time algorithms for some minimum latency problems. *Information Processing Letters*, 75:225–229, 2000.

B.Y. Wu, Z.-N. Huang, and F.-J. Zhan. Exact algorithms for the minimum latency problem. *Information Processing Letters*, 92:303–309, 2004.