Les Cahiers du GERAD ISSN: 0711–2440

A General Variable Neighborhood Search for the One-Commodity Pickup-and-Delivery Travelling Salesman Problem

S. Hanafi, A. Ilić,
D. Urošević, N. Mladenović

G–2011–33

June 2011

A General Variable Neighborhood Search for the One-Commodity Pickup-and-Delivery Travelling Salesman Problem

Saïd Hanafi

LAMIH - Université de Valenciennes
ISTV 2 Le Mont Houy
59313 Valenciennes Cedex 9, France
said.hanafi@univ-valenciennes.fr

Aleksandar Ilić

Faculty of Mathematics and Sciences
University of Niš
Niš, Serbia
aleksandari@gmail.com

Dragan Urošević

Mathematical Institute
Serbian Academy of Science
Belgrade, Serbia
draganu@mi.sanu.ac.rs

Nenad Mladenović

GERAD & School of Mathematics
Brunel University
Uxbridge UB8 3PH, UK
nenad.mladenovic@brunel.ac.uk

June 2011

Les Cahiers du GERAD

G–2011–33

Abstract

We present a Variable Neighborhood Search approach to solving the one-commodity pickup-and-delivery travelling salesman problem. It is characterized by a set of customers, each of them supplying (as pickup customers) or demanding (as delivery customers) a given amount of a single product. A vehicle with a given capacity starts at the depot and must visit each customer once only and, the vehicle's capacity must not be exceeded. The objective is to minimize the total length of the tour. Thus, the problem considered contains the checking of existence of feasible travelling salesman's tour and the design of the optimal travelling salesman's tour, both being NP-hard. We adapt a collection of neighborhood structures which are mainly used for solving the classical travelling salesman problem: k–opt, double-bridge and insertion operators. Using a binary indexed tree, we efficiently update the data structures for feasibility checking in these neighborhoods. Our extensive computational analysis shows that the proposed variable neighborhood search based heuristics outperforms the best-known algorithms in terms of both the solution quality and computational efforts. Moreover, we improve the best-known solution of all benchmark instances from the literature (with 100 to 500 customers). We are also able to solve instances with up to 1000 customers.

Key Words: Combinatorial Optimization; Metaheuristics; Variable neighborhood search; Pickup-and-delivery Travelling salesman problem.

Résumé

Nous présentons une approche de recherche à voisinage variable pour résoudre le problème du voyageur de commerce avec une commodité chargement et déchargement. Il est caractérisé par un ensemble d'usagers, chacun d'entre eux offrant (comme usager de chargement) ou demandant (comme usager de déchargement) une quantité donnée d'un seul produit. Un véhicule avec une capacité donnée part du dépôt et doit visiter chaque usager une fois seulement et la capacité ne peut être dépassée. L'objectif est de minimiser la longueur totale du trajet. Donc le problème considéré contient la vérification d'existence d'un trajet admissible pour le problème du voyageur de commerce et la détermination d'un trajet optimal pour le voyageur de commerce tous deux étant NP-difficiles. Nous adaptons une collection de structures de voisinage qui sont principalement utilisées pour résoudre le problème du voyageur de commerce classique : k–opt, double-pont et opérateur d'insertion. À l'aide d'un arbre binaire indexé, nous mettons à jour efficacement les structures de données pour vérifier la faisabilité dans ce voisinage. Des expériences de calcul extensives montrent que l'algorithme VNS proposé donne de meilleurs résultats que les algorithmes plus connus, tant en terme de la qualité de la solution que du temps de calcul. De plus, nous améliorons la meilleure solution de tous les problèmes tests de la littérature (avec de 100 à 500 usagers). Nous sommes aussi capables de résoudre des instances avec jusqu'à 1000 usagers.

Acknowledgments: The present research work has been supported by Research Grant 174010 of the Serbian Ministry of Science and Environmental Protection, International Campus on Safety and Intermodality in Transportation, the Nord-Pas-de-Calais Region, the European Community, the Regional Delegation for Research and Technology, the Ministry of Higher Education and Research, and the National Center for Scientific Research. The authors gratefully acknowledge the support of these institutions. We are grateful to Inmaculada Rodrǵuez-Martń and Juan José Salazar-González for providing us with the executable version of the algorithm proposed in Hernández–Pérez et al. (2009).

1 Introduction

The one-commodity pickup-and-delivery travelling salesman problem (1-PDTSP for short) was introduced by Hernández-Pérez and Salazar-Gonzales (2004a), although similar extended variants of the travelling salesman problem (TSP) were suggested earlier (see, e.g., Chalasani and Motwani (1999), Gendreau et al. (1999), Mosheiov (1994), Savelsbergh and Sol (1995)). Surveys may be found in Parragh et al. (2008a) and Parragh et al. (2008b).

Problem description. 1-PDTSP is a routing problem which generalizes the classical TSP as follows. A set of location is given, with the travel distances or costs between them; one specific location is considered to be the vehicle depot, while all the others are identified with customers. These customers are divided into delivery customers and pickup customers, according to the type of service they require. A unique commodity or product has to be transported from some customers to others and each delivery/pickup customer requires/provides a given amount of this commodity. A vehicle with a given capacity starts and ends at the depot and must visit each customer only once. The vehicle capacity must not be exceeded after visiting a pickup customer and any delivery customer can be served so long as the load on the vehicle is no less than its demand. The 1-PDTSP consists of finding a minimum length Hamiltonian route for the vehicle which satisfies all the customer requirements. It is not assumed that the vehicle leaves either empty or fully loaded from the depot, and the initial load of the vehicle also has to be determined.

The 1-PDTSP has several applications in routing a single commodity through a circular network in a graph connecting different sources and destinations. A real-world application of the 1-PDTSP is shown by Anily and Bramel (1999) in the context of inventory reposition. Consider a set of retailers owned by the same firm and located at different sites in a state. At a given moment, due to the random nature of the demand, some retailers may have an excess of inventory, while others are in need of additional stock. If the firm decides to transfer inventory from the first group of retailers to the second one, then determining the cheapest Hamiltonian route to serve all the retailers with a capacitated vehicle is exactly the 1-PDTSP. For example, the customers can be branches of a bank in an area providing or requiring a known amount of money (the product), and the depot is the main branch of the bank. Clearly, this is a simple variant of a more realistic problem where several commodities could be considered or several capacitated vehicles used in transportation, but it is still an interesting problem in its own right.

Notation. We now introduce the notation that will be used throughout this article. Let $G = (V, E)$ be a complete and undirected graph, with vertex set $V = \{1, 2, \ldots, n\}$. Vertex 1 represents the depot. The other vertices from 2 to n represent the customers, each with an associated non-zero demand q_i. If $q_i > 0$, i is a pickup customer; otherwise, it is a delivery customer. The depot can be considered as a customer with demand $q_1 = -\sum_{i=2}^{n} q_i$. If $q_1 > 0$, this means the initial load of the vehicle; otherwise q_1 represents the final load after visiting all customers. The travel cost $c(i, j)$ for each edge $(i, j) \in E$ is given. It can be assumed that the travel costs between locations are proportional to the distance. The capacity of the vehicle is represented by $Q > 0$. A vehicle with capacity Q must visit each customer only once, and its capacity must not be exceeded after visiting a pickup customer and its load must be greater than (or equal to) the demand of the delivery customer before visiting it. The 1-PDTSP should determine a tour starting and ending at the depot and having minimum length. Note that typically it holds

$$\max_{i\in V} |q_i| \leq Q \leq \max\{\sum_{i\in V,\ q_i>0} q_i, -\sum_{i\in V,\ q_i<0} q_i\}.$$

The left inequality means that no feasible tour can be found if the vehicle capacity is less than the maximum customer demand, while the second one makes this problem different from the standard TSP.

Previous work. The 1-PDTSP is NP-hard since it coincides with the Travelling Salesman Problem when the vehicle capacity is large enough. Even more, the problem of checking the existence of a feasible solution is NP-complete in the strong sense (Hernández–Pérez (2004)).

Hernández-Pérez and Salazar-González (2004b) describe an exact branch-and-cut algorithm able to solve instances with up to 60 customers. The same authors propose in Hernández–Pérez and Salazar-González

(2004a) two heuristic approaches to deal with larger instances. The first heuristic is a simple local search developed to provide an initial upper bound for the branch-and-cut algorithm. The second approach is a more elaborated algorithm based on 'incomplete optimization'. That is, the branch-and-cut algorithm is applied to a restricted search space obtained by considering only a subset of variables, associated with promising edges of the graph. Moreover, the branch-and-cut execution is truncated by imposing a limit to the number of levels in the search tree exploration. A primal heuristic is also embedded in the branch-and-cut to build feasible integer solutions from the information given by the fractional solutions. This heuristic is periodically applied during the tree search process.

The algorithm proposed in Hernández–Pérez et al. (2009) for solving the 1-PDTSP is a hybrid algorithm which combines the Greedy Randomized Adaptive Search Procedure (GRASP) and Variable Neighborhood Descent (VND) paradigms. The GRASP metaheuristic is a multi-start procedure, consisting of a loop embedding, a construction phase and a local search phase. The best overall solution is kept as the final result. The two local search operators are adaptations of the 2–opt and 3–opt edge-exchange operators for TSP (see e.g., Johnson and McGeoch (1997)). The GRASP loop is iterated until the stopping condition is met (a combination of the total number of iterations and computation time). Then, a post-optimization phase consists of a second VND which starts from the best solution found so far and consists of vertex-exchange neighborhoods: forward and backward insertion operators (shifting only one customer from the current position in the tour to any other position).

In Hernández–Pérez and Salazar-González (2007), the authors pointed out a close connection between the 1-PDTSP and the classical capacitated vehicle routing problem (CVRP), where a homogeneous capacitated vehicle fleet located in a depot must collect a product from a set of pickup customers. These authors presented new inequalities for the 1-PDTSP adapted from recent inequalities for the CVRP and these inequalities have been implemented in a branch-and-cut framework to solve to optimality the 1-PDTSP.

Recently Zhao et al. (2009a) proposed a genetic algorithm (GA) which on average gave better results than the GRASP/VND heuristic. The authors in fact suggested a hybrid heuristic which within standard GA operators uses several local searches and Ant colony ideas. After initializing the parameters used in the algorithm, the genetic algorithm starts from a random population of individuals (feasible tours), followed by 2–opt local optimization and iterates for some fixed number of generations. A new pheromone-based method for a crossover operator was described, which utilizes both global and local information (edge length, adjacency relations, demands of customers) to construct an offspring. In the proposed GA, the pheromone trails are updated on the basis of the globally best solution. They used 3 vertex exchange as the mutation operator as follows: select three customers at random and then exchange their positions in the tour; the best feasible one among 5 possible is chosen. Furthermore, the authors proposed a better iterative probabilistic method for constructing the initial solution, based on the nearest neighbors to each customer. A similar heuristic is developed in Zhao et al. (2009b) for another similar PDTSP version.

Chalasani and Motwani 1999 have studied k-delivery TSP. This problem can be seen as a special case of the 1-PDTSP, where both delivery and pickup amount are equal to one unit. In the capacitated dial-a-ride problem (CDARP) (also named the single-vehicle many-to-many dial-a-ride problem), there is a one-to-one correspondence between pickup customers and delivery customers, and the vehicle should move one unit of a commodity from its origin to its destination with a limited capacity Q (see Guan (1998)). Another related problem is the TSP with backhauls (TSPB) (see Gendreau et al. (1997)), where an uncapacitated vehicle must visit all the delivery customers before visiting a pickup customer.

The rest of this present paper is organized as follows. In Section 2 we give rules of general variable neighborhood search heuristics for solving the 1-PDTSP. In Section 3 we describe our data structure and show the complexity of updating the feasibility of the solution when different neighborhood structures are used. Section 4 consists of computational results on the benchmark and some new large instances. Section 5 concludes the paper.

2 General Variable Neighborhood Search for 1-PDTSP

In this section we first give the steps of the General Variable Neighborhood search (GVNS) metaheuristic and then describe the algorithmic components which we have designed for solving 1-PDTSP: the initialization process, the three neighborhood structures used as local search procedures and the two neighborhoods used in the shaking step.

2.1 General Variable Neighborhood Search

Variable Neighborhood Search (VNS) (Mladenović and Hansen (1997)) is a metaheuristic, or framework, for building heuristics whose basic idea is a systematic change of neighborhood structures within the local search algorithm. To construct different neighborhood structures and to perform a systematic search, one needs to supply the solution space with some (quasi)-metrics and then induce neighborhoods from them. Different neighborhood structures can be exploited in both deterministic and stochastic ways. The basic VNS combines both approaches: a point from the k^{th} neighborhood is taken at random from where the deterministic local search algorithm starts. If a better solution is found, the new incumbent is obtained and the search is re-centered around it. Otherwise, a random point is generated from the neighborhood $k+1$, etc (for recent surveys of VNS, see Hansen et al. (2008), Hansen et al. (2010) and for its convergence properties see Brimberg et al. (2010)).

General VNS (GVNS) is an extended version of the basic VNS. It simply uses more than one neighborhood in a local search. Such a local search is called a Variable Neighborhood Descent (VND). The steps of the general VNS and VND are given in Algorithm 8 and Algorithm 2 below.

Algorithm 1: Steps of the general VNS

```
  Function GVNS ($x, \ell_{max}, k_{max}, t_{max}$)
1 repeat
2     $k \leftarrow 1$;
3     repeat
4         $x' \leftarrow$ Shake($x, k$);
5         $x'' \leftarrow$ Seq-VND($x', \ell_{max}$) / Mix-VND($x', \ell_{max}$) ;
6         $k, x \leftarrow$ NeighborhoodChange($x, x'', k$);
      until $k = k_{max}$
7     $t \leftarrow$ CpuTime();
  until $t > t_{max}$
8 return $x$.
```

We now detail the steps of the GVNS. Let us denote by $\mathcal{N}_k$, for $k = 1, \ldots, k_{max}$, a finite set of pre-selected neighborhood structures, and with $\mathcal{N}_k(x)$ the set of solutions in the k^{th} neighborhood of x. These neighborhoods are used within VNS for diversification purposes, i.e., in the Shaking step.

Sequential VND (`Seq-VND`). We also denote by $N_\ell, \ell = 1, \ldots, \ell_{max}$, neighborhood structures used within VND. Here we first discuss the general difference between the two basic strategies in VND: sequential and nested. The steps of Sequential VND are presented in Algorithm 2. We call it sequential because the neighborhood structures are explored one by one in the given sequence. Note that each local search, and thus the VND, could use *first improvement* (make a move the first time an improvement in the neighborhood has been observed) or *best improvement* strategy (move to the best solution in the neighborhood). The latter is also known as *steepest descent*. For all variants of our VND that will be discussed below, we use the first improvement approach (for the discussion regarding differences between these two strategies in solving the travelling salesman problem by 2–opt, see Hansen and Mladenović (2006)).

The function `NeighborhoodChange()` is common for both VNS and VND. Its pseudo-code is given at Algorithm 3. It compares the value $f(x')$ with the incumbent value $f(x)$ obtained in the neighborhood k for VNS, or ℓ for the VND. If an improvement is obtained, k (or ℓ) is returned to its initial value and the new incumbent updated (line 2). Otherwise, the next neighborhood is considered (line 3).

Algorithm 2: Sequential Variable Neighborhood Descent

```
  Function Seq-VND($x, \ell_{max}$)
1 $\ell \leftarrow 1$                                   // Neighborhood counter
2 repeat
3     $i \leftarrow 0$                                  // Neighbor counter
4     repeat
5         $i \leftarrow i + 1$
6         $x' \leftarrow arg\min\{f(x), f(x_i)\}, x_i \in N_\ell(x)$ // Compare
      until ($f(x') < f(x)$ or $i = |N_\ell(x)|$)

7     $\ell, x \leftarrow$ NeighborhoodChange ($x, x', \ell$);    // Neighborhood change
  until $\ell = \ell_{max}$
8 return $x$.
```

Algorithm 3: Neighborhood change or Move or not function

```
  Procedure NeighborhoodChange ($x, x', k$)
1 if $f(x') < f(x)$ then
2     $x \leftarrow x'; k \leftarrow 1$;    // Make a move
  else
3     $k \leftarrow k + 1$;         // Next neighborhood
```

Most local search heuristics in their descent phase use very few neighborhoods (usually one or two, i.e., $\ell_{max} \leq 2$). Note that the final solution of Seq-VND should be a local minimum with respect to all ℓ_{max} neighborhoods. Hence the chances of reaching a global minimum are greater than with a single neighborhood structure. The total size of Seq-VND is obviously equal to the union of all neighborhoods used. Thus, if neighborhoods are pairwise disjoint (with no common element in any two neighborhoods) then the following holds

$$|N_{\texttt{Seq-VND}}(x)| = \sum_{\ell=1}^{\ell_{max}} |N_\ell(x)|, \; x \in X.$$

Nested and Mixed nested VND. Beside this sequential order of neighborhood structures in the VND above, one can develop a *nested* (Nest-VND for short) or mixed-nested VND (Mix-VND) according to Ilić *et al.* (2010). Assume that we define two neighborhood structures ($\ell_{max} = 2$). In the nested VND we in fact perform local search with respect to the first neighborhood in any point of the second. Thus, the cardinality of the neighborhood obtained with the nested VND is clearly the product of the cardinalities of the neighborhoods included, i.e.,

$$|\mathcal{N}_{\texttt{Nest-VND}}(x)| = \prod_{\ell=1}^{\ell_{max}} |\mathcal{N}_\ell(x)|, \; x \in X.$$

Therefore, the pure Nest-VND neighborhood is much larger than the sequential one. This implies that the number of local minima with respect to Nest-VND will be much smaller than the number of local minima with respect to Seq-VND.

In the Mix-VND after exploring b (a parameter) neighborhoods, we switch from a nested to a sequential strategy. We can interrupt nesting at some level b' ($b' = \ell_{max} - b + 1, \; 0 \leq b \leq \ell_{max}$) and continue with the list of the remaining neighborhoods in sequential manner. If $b = 0$, we get Seq-VND. If $b = \ell_{max}$ we get Nest-VND. Since nested VND intensifies the search in a deterministic way, the boost parameter b may be seen as a balance between intensification and diversification in deterministic local search with several neighborhoods. Its cardinality is clearly

$$|N_{\texttt{Mix-VND}}(x)| = \sum_{\ell=1}^{\ell_{max}-b} |N_\ell(x)| \times \prod_{\ell=\ell_{max}-b+1}^{\ell_{max}} |\mathcal{N}_\ell(x)|, \qquad x \in X.$$

Algorithm 4: Nested Variable Neighborhood Descent

```
   Function Nest-VND $(x, x', k)$
1  Make an order of all the $\ell_{max} \geq 2$ neighborhoods that will be used in the search
2  Find an initial solution $x$; let $x_{opt} = x$, $f_{opt} = f(x)$
3  Set $\ell = \ell_{max}$
4  repeat
5      if all solutions from $\ell$ neighborhood are visited then $\ell = \ell + 1$
6      if there is any non visited solution $x_\ell \in N_\ell(x)$ and $\ell \geq 2$ then $x_{cur} = x_\ell$, $\ell = \ell - 1$
7      if $\ell = 1$ then
8          Find objective function value $f = f(x_{cur})$
9          if $f < f_{opt}$ then $x_{opt} = x_{cur}$, $f_{opt} = f_{cur}$
   until $\ell = \ell_{max} + 1$ (i.e., until there are no more points in the last neighborhood)
```

Shaking. The Shaking step is in common for both basic VNS and GVNS. It generates randomly a solution from the k-th neighborhood of the current solution x, or performs sequentially k random moves.

In the following subsections, we present the main features of our adaptation of GVNS metaheuristic to the 1-DPTSP.

2.2 Initial Solution for PDTSP

The solution space of 1-PDTSP consists of all possible tours starting from the depot. A tour will be represented as a permutation of $\{1, \ldots, n\}$. Many of these tours are infeasible since the amount of the commodity collected from consecutive pickup customers on the tour can exceed the residual capacity of the vehicle or the amount of commodity requested by consecutive delivery customers can exceed the maximum load of the vehicle. We start from a tour which may be infeasible. However, once a feasible tour is reached, we restrict our search to the subspace of feasible tours.

Let $x = (x_1, x_2, \ldots, x_n)$ denote a tour which passes through each customer exactly once with $x_1 = 1$ and let $L_i(x)$ be the load of the vehicle after visiting the i-th customer in the tour x,

$$L_i(x) = L_{i-1}(x) + q_{x_i} \qquad \text{and} \qquad L_1(x) = q_{x_1}.$$

Then, as shown in Hernández–Pérez and Salazar-González (2004a), x is a feasible tour of 1-PDTSP if and only if

$$\max_{i \in V} L_i(x) - \min_{i \in V} L_i(x) \leq Q. \tag{1}$$

We use a method from Zhao et al. (2009a) for generating the initial solution, since it gives better results on average than the algorithm proposed in Hernández–Pérez et al. (2009). This constructive procedure starts by randomly choosing the first customer x_2. The next customer x_{i+1} added to the current sub-tour T is chosen from a static data structure which includes $cl = 20$ closest customers from the last inserted customer x_i. We consider only feasible sub-tours without violating the capacity constraints to deliver or to collect the demand with respect to the customer x_{i+1}. Among these cl closest customers, we search for the customers who can be feasibly added at the end of the tour T and who have not yet been visited – and select the customer with the largest demand. If such a customer does not exist, we search for all customers who have not appeared in the sub-tour T and let S be the set of customers who can be feasibly added to the sub-tour. We select the nearest customer from S with a probability of 0.9, or select a random customer from S with a probability of 0.1. Finally, if there are no customers who can be added ($S = \emptyset$), we add a random customer and continue. Note that this algorithm can end with an infeasible solution.

2.3 Neighborhood Structures for the 1-PDTSP

In a local search algorithm a neighborhood structure is introduced to generate moves from one solution to another. The final solution provided by a local search is a local optimum which can not be improved by using the neighborhood structure under consideration. Fundamental neighborhood structures for the TSP which are based on edge exchanges and node insertion moves can also be extended efficiently for 1-PDTSP. Classical heuristics of this type are the k-exchanges (Lin and Kernighan (1973)), also called $k--opt$, which are successfully used in local searches for assignment, routing and scheduling problems (see, e.g. Gamboa et al. (2006), Laporte (1992), Rego et al. (2010)).

The 2–opt procedure is the simplest method; it removes two edges from the tour and reconnects the two paths created. There is only one way to reconnect two paths and still have a valid tour. We perform this move only if the new tour will be shorter and we continue removing and reconnecting the tour until no 2–opt improvements can be found. By the so called *neighbor-list implementation* of 2–opt (see, e.g., Hansen and Mladenović (2006)), we sort all customers in non-decreasing order of their distances in the preprocessing step, i.e., we construct the ranked matrix for them. For the selected edge $e_i = (x_i, x_{i+1})$, we search for another edge $e_j = (x_j, x_{j+1})$ such that

$$c(x_i, x_{i+1}) + c(x_j, x_{j+1}) > c(x_i, x_j) + c(x_{i+1}, x_{j+1}).$$

For the fixed edge e_i, the customers x_j are processed in non-decreasing order according to their distances from the customer x_i (and similarly customers x_{j+1} are processed according to their distances from the customer x_{i+1}). Clearly, we can break the processing of vertices if $c(x_i, x_j) > c(x_i, x_{i+1})$ (and similarly we can break the processing of vertices according to their distances from x_{i+1} if $c(x_{i+1}, x_{j+1}) > c(x_{i+1}, x_i)$). In other words, the neighbor-list implementation of the 2–opt significantly reduces the number of pairwise choices of edges e_i and e_j.

The k–opt neighborhood is a generalization of the 2–opt. It drops some k edges and adds k new edges in order to reconstruct a new tour. There are several possible ways to reconstruct a feasible tour after deleting k edges. In addition, the number of neighboring solutions in a k–opt neighborhood is $O(n^k)$ (Johnson and McGeoch (1997)). Hence, exploring complete k–opt neighboring solutions is costly. This is why restricted k–opts are generally used:

(i) *Insertion* is a special case of 3–opt where each customer is inserted between any other two customers. Its cardinality is $O(n^2)$;

(ii) *Or–opt* (Or (1976)) is also a special case of 3–0opt. As well as inserting one customer, by Or–opt each two or three consecutive customers on the tour are inserted between any other two customers. Its cardinality is also $O(n^2)$;

(iii) *Swap* (interchange) is a special case of 4–opt where 2 customers exchange places in the tour. Its cardinality is obviously $O(n^2)$;

(iv) *Double bridge* (Johnson and McGeoch (1996)) is a special case of 4–opt as well (see Figure 1 (a)). It is very important for some combinatorial problems, since it keeps the orientation of the tour.

In this paper, beside 2–opt, we use Insertion, 3–opt and double-bridge neighborhood structures within the General VNS framework; see Figure 1. In addition, Insertion neighborhood is split into two disjoint subsets: forward and backward insertion.

The *forward insertion* move tries to find a shorter tour by moving a customer from its current position i in the tour, to some position j with $j > i$. This implies that all customers in positions $i+1, i+2, \ldots, j$ have to be shifted backwards one position. Customers in positions 1 to $i-1$ and $j+1$ to n remain unchanged.

The *backward insertion* operator works in a similar way, but this time the selected customer at position i, is moved to a position j in the tour, such that $j < i$, and intermediate customers are shifted forward one position.

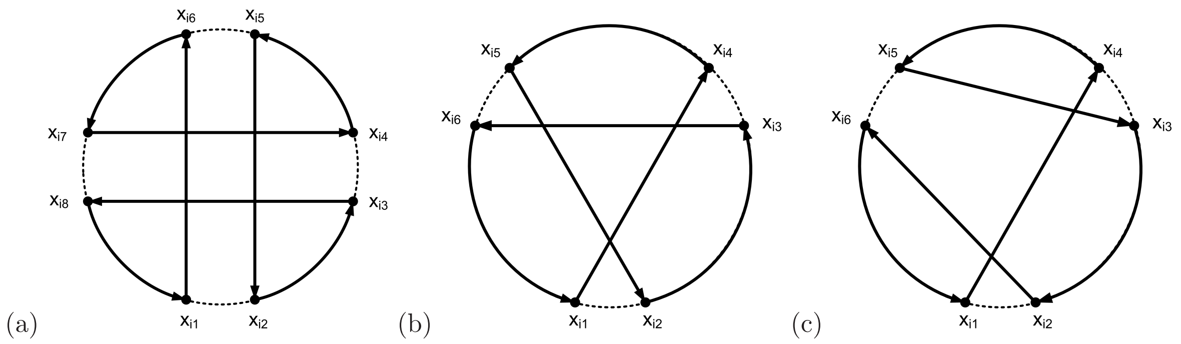

Figure 1: Double–bridge (a) and 3–opt without change of direction (b) and with change of direction (c)

2.4 Local Search and Shaking

A complete local search is organized as Variable neighborhood descent through a 2–opt neighborhood (N_1), forward and backward insertion neighborhoods (N_2 and N_3), i.e., a parameter ℓ_{max} from Algorithm 2 has the value 3. We call it `Seq-VND`. Another local search is designed as Mixed-nested VND (see also Ilić *et al.* (2010) for details): let x be the current solution; we take solution x' from 3–opt or double–bridge neighborhoods of x at random and then apply sequential VND (which consists of 2–opt, forward and backward insertion neighborhoods) starting from solution x'. If such an obtained solution is of better quality than the incumbent x, we move to it and re-centre the search around it ($x \leftarrow x'$). Otherwise we repeat the complete procedure, i.e., we again choose a new solution at random followed by the sequential VND. The maximum number of such unsuccessful trials is set at 200. Therefore, this local search is not fully deterministic.

Shaking is performed by using two well known neighborhoods designed for solving TSP: 3–opt and double–bridge (see Figure 1). We get a random point from $\mathcal{N}_k$ neighborhood by performing k_{3--opt} 3–opt moves and k_{db} double–bridge moves, where k_{3--opt} is a randomly chosen parameter from $[k/2, k]$ and $k_{db} = k - k_{3--opt}$. The 3–opt and double–bridge moves are performed such that feasibility is kept (assuming that the incumbent solution is feasible). Otherwise, a random point from 3–opt or from double–bridge neighborhoods is generated.

3 An Efficient Implementation of GVNS for 1-PDTSP

From the point of view of implementation, the main difference between the 1-PDTSP and TSP is that in the first problem not any permutation of customers represents the feasible tour. Therefore, we need to design a data structure which will allow us to obtain efficiently tours which not only are shorter, but are also feasible. In this section we describe the computer implementation of feasibility checking on all neighborhood structures used within our VND, i.e., within our General VNS.

3.1 2–opt with Feasibility Checking

As mentioned in the previous section, we use what is called the neighbor-list implementation of 2–opt (Johnson and McGeoch (1997)). In addition, for each solution from the 2–opt neighborhood of the current one, we need to check if it is feasible. In order to do this efficiently, we propose here the use of a *binary indexed tree* (BIT) data structure. This data structure will allow us to check the feasibility of the neighbor tour in $O(\log n)$ (instead of $O(n)$ if the usual data structure is used). If a better solution is found, we need to construct a new BIT structure in $O(n \log n)$ time.

BIT is an efficient data structure introduced by Fenwick (1994) for maintaining the cumulative frequencies (or partial summations). The basic idea is that each integer can be represented as the sum of certain powers of two and therefore a cumulative frequency can be also represented as the sum of certain sets of cumulative sub-frequencies. The operations to access this data structure are based on the binary coding of the index.

Let L be an array of n elements. The binary indexed tree supports the following basic operations:

- for a given value v and index i, add v to the element L_i, $1 \leq i \leq n$;
- for a given interval $[i, j]$, find maximum/minimum among values $L_i, L_{i+1}, \ldots, L_j$, $1 \leq i \leq j \leq n$.

We store the data structure in the array $Tree$ (see Algorithm 5 and Algorithm 6 for the maximum case). The structure is space-efficient in the sense that it needs the same amount of storage as merely a simple array of n elements. For vehicle loads we will use a binary indexed tree data structure to compute the maximum and minimum values in the intervals of vehicle loads. The structure $Tree$ is a complete binary tree with the root node 1. Its leafs correspond to the elements from the array L, moving from left to right in the last level. Therefore, the elements of the array L are stored at the positions which start from 2^p to $2^p + n - 1$, where p is the depth of the binary tree defined as the smallest integer such that $2^p \geq n$. The internal nodes of the $Tree$ store the cumulative values (sum, max, min, ...) of the leafs in the subtrees rooted at these nodes. This implies that the value of the internal node i is just the cumulative value of its two children. The parent of the node i is $\lfloor \frac{i}{2} \rfloor$, while the left and the right children of the node i are $2i$ and $2i + 1$, respectively. By definition, it follows that the number of nodes in BIT is at most $2n$, while the depth is $\lceil \log_2 n \rceil$.

For the updating procedure, we simply need to traverse the vertices from the leaf to the root and update the values in the parent vertices. For the query procedure, we consider the parents of the left (i) and right (j) vertices and the intervals which they cover and then iteratively calculate max/min in $[i, j]$. It can easily be seen that the max/min operators can be replaced by any distributive function such as sum, product, etc. The binary indexed tree is updated after setting the element L_i to the new value v as described in Algorithm 5.

In the algorithms we use fast binary operators. The bitwise exclusive OR operator (XOR) used in these algorithms compares each bit of its first operand to the corresponding bit of its second operand. If one bit is 0 and the other bit is 1, the corresponding result bit is set to 1. Otherwise, the corresponding result bit is set to 0. Furthermore, instead of using **div** operator for integer division by 2, we can use the bitwise shift left operator **shl**.

Algorithm 5: Updating the binary indexed tree, after setting the element L_i to the new value v

```
1 Input: The value $v$, element index $i$ and $p = \max\{s : 2^s < n\}$.
2 $i \leftarrow i + 2^{p+1} - 1$;
3 $Tree[i] \leftarrow v$;
4 while $i > 0$ do
5     $Tree[i$ div $2] \leftarrow \max(Tree[i], Tree[i$ xor $1])$;
6     $i \leftarrow i$ div $2$;
```

Algorithm 6: Calculating the maximum value in interval $L_i, L_{i+1}, \ldots, L_j$

```
1 Input: The parameters $i$ and $j$ are extreme values of interval $[i, j]$ and $p = max\{s : 2^s < n\}$.
2 Output: The maximum value among elements $L_i, L_{i+1}, \ldots, L_j$.
3 $i \leftarrow i + 2^{p+1} - 2$; $j \leftarrow j + 2^{p+1}$;
4 $v_{max} \leftarrow -1$;
5 while ($i$ div 2) $\neq$ ($j$ div 2) do
6     if ($i$ and 1) $= 0$ then $v_{max} \leftarrow \max(v_{max}, Tree[i + 1])$;
7     $i \leftarrow i$ div 2;
8     if ($j$ and 1) $\neq 0$ then $v_{max} \leftarrow \max(v_{max}, Tree[j - 1])$;
9     $j \leftarrow j$ div 2;
10 return $v_{max}$.
```

Property 3.1 *Updating the binary index tree after setting the element L_i to the new value is executed in $O(\log n)$.*

Proof. First we need to update the corresponding node in the tree that keeps the value L_i, then the iteratively traverse parents (i, $parent[i]$, $parent[parent[i]]$, ...) and update the cumulative values in these nodes. Since the depth of the tree is at most $\lceil \log_2 n \rceil$, the overall complexity $O(\log n)$. □

Property 3.2 *Calculating the maximum value in interval $[L_i, L_j], j > i$, is executed in $O(\log n)$ time.*

Proof. To begin with, we need to use the offset $2^p - 1$ in order to get the nodes in BIT which correspond to the indices i and j. Value i will represent the leftmost subtree, while the value j will represent the rightmost subtree that has a nonempty intersection of the given interval. Nodes i and j will always be at the same level of the binary indexed tree, and we will update the global cumulative value (the maximum function in Algorithm 6) as long as the parents of i and j are different. If Node i is the left child of its parent, then the whole right subtree rooted at $i+1$ is contained in the given interval and we need to update the global cumulative value with $Tree[i+1]$. Similarly, if Node j is the right child of its parent, then the whole left subtree rooted at $j-1$ is contained in the given interval and we need to update the global cumulative value with $Tree[j-1]$. Since the query algorithm visits exactly two nodes per level of the binary tree, calculating the maximum value in the interval $[L_i, L_j]$ takes $O(\log n)$ time in total. □

Property 3.3 *Checking the feasibility of the 2–opt move for 1-PDTSP with BIT structure is in $O(\log n)$.*

Proof. Let $x = (x_1, \ldots, x_n)$ be a current tour and $L = (L_1, \ldots, L_n)$ its associated load vector (where $L_i = L_i(x)$ is the load of vehicle after visiting the i-th customer in the tour x). We will use Equation (1) for checking the feasibility of the solution. Let $Tree_{max}$ and $Tree_{min}$ be the binary indexed structures for computing the maximum and minimum vehicle loads for each city. After performing the 2–opt move with attributes i and j associated with the edges $e_i = (x_i, x_{i+1})$ and $e_j = (x_j, x_{j+1})$ with $1 \le i < j \le n$, the new tour x' looks like

$$x_1 \to x_2 \to \ldots \to x_i \to x_j \to x_{j-1} \to \ldots \to x_{i+1} \to x_{j+1} \to x_{j+2} \to \ldots \to x_{n-1} \to x_n \to x_1.$$

or as

$$x'_k = \begin{cases} x_k & \text{for } \; k = 1, \ldots, i, \\ x_{i+j-k+1} & \text{for } \; k = i+1, \ldots, j \\ x_k & \text{for } \; k = j+1, \ldots, n. \end{cases}$$

Thus, only the vehicle loads for the customers $x_{i+1}, x_{i+2}, \ldots, x_j$ are changed. More precisely, the new load L' associated with the new solution x' is computed as follows:

$$L'_k = \begin{cases} L_k & \text{for } \; k = 1, \ldots, i, \\ L_i + \sum_{h=i+1}^{k} q_{x_{i+j-h+1}} & \text{for } \; k = i+1, \ldots, j \\ L_k & \text{for } \; k = j+1, \ldots, n. \end{cases}$$

Since $L_0 = 0$ and $\sum_{i=1}^{n} q_{x_i} = 0$, it is easy to see that we have

$$L_i + \sum_{h=i+1}^{k} q_{x_{i+j-h+1}} = L_i + L_j - L_{k-1} \qquad \text{for} \qquad k = i+1, i+2, \ldots, j-1, j.$$

Therefore, in order to find the new minimum vehicle load, we need to find the maximum load among $L_{j-1}, L_{j-2}, \ldots, L_i$ and subtract this number from $L_i + L_j$. This can be computed in $O(\log n)$ time, using the $Tree_{min}$ structure for the interval $[i, j-1]$ (see Properties 3.1 and 3.2). Analogously, we calculate the new maximum vehicle load and check the feasibility of the new tour x'. □

3.2 Illustrative Example

For illustration, we consider the small example n20q10B.tsp from the benchmark instances of 1-PDTSP (see Section 4) with $n = 20$ nodes and capacity $Q = 10$. The optimal tour $x^\star$ and a suboptimal tour x with their associated loads are:

$x^\star = (1, 11, 17, 14, 19, 4, 8, 20, 15, 18, 12, 6,\ \mathbf{16,\ 10,\ 2,}\ 9, 13, 5, 7, 3, 1)$,
$x = (1, 11, 17, 14, 19, 4, 8, 20, 15, 18, 12, 6,\ \mathbf{2,\ 10,\ 16,}\ 9, 13, 5, 7, 3, 1)$,
$L(x^\star) = (8, 3, 8, 0, 10, 7, 8, 1, 7, 1, 3, 3,\ \mathbf{4,\ 9,\ 6,}\ 7, 3, 4, 0, 0)$,
$L(x) = (8, 3, 8, 0, 10, 7, 8, 1, 7, 1, 3, 3,\ \mathbf{0,\ 5,\ 6,}\ 7, 3, 4, 0, 0)$.

Tours $x^\star$ and x are presented in Figure 2, and their associated BIT structures for the loads $L(x^\star)$ and $L(x)$ are presented in Figure 3. Observe that the optimal tour $x^\star$ is derived from the suboptimal x by one 2–opt move by considering the edges $(6, 16)$ and $(2, 9)$, i.e. by reversing the part 16, 10, 2. This 2–opt move changes the values in three leaves of the binary indexed tree (as shown in Figure 3) which corresponds to Nodes 13, 14 and 15, and their parents.

Also note that the optimal tour crosses itself, which is not possible for the Euclidean TSP optimal tour.

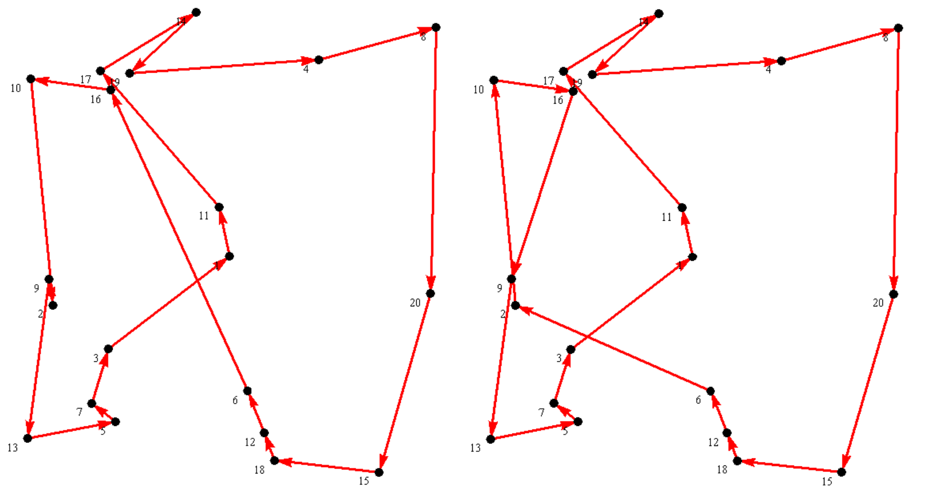

Figure 2: Optimal and suboptimal tour

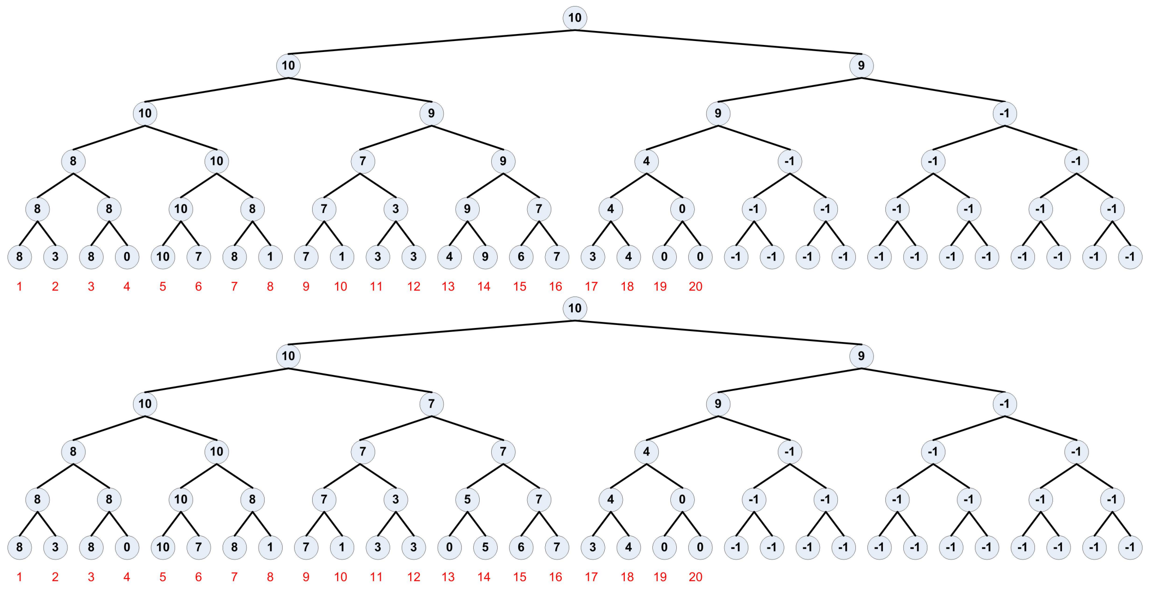

Figure 3: Binary indexed trees for two tours

Insertion feasibility checks. For the feasibility checking of Insertion (forward and backward) moves, the binary indexed structure is not necessary.

Property 3.4 *Feasibility checking of insertion moves for 1-PDTSP is in $O(1)$ time.*

Proof. We traverse the customers i from 1 to n in that order. After moving the customer i to the position $j > i$, we have the following loads

$$L_1, L_2, \ldots, L_{i-1}, L_{i+1} - q_{x_i}, L_{i+2} - q_{x_i}, \ldots, L_{j-1} - q_{x_i}, L_j - q_{x_i}, L_j, L_{j+1}, \ldots, L_n.$$

Therefore, we need to maintain the minimum and maximum values of $L_{i+1}, L_{i+2}, \ldots, L_j$, which can be updated in $O(1)$ time after traversing from position j to $j+1$. □

The overall complexity for an Insertion local search is $O(n^2)$.

3.3 Maintaining the Feasibility of 3–opt and Double Bridge Moves

Once feasibility is reached, one would like to keep it in both the local search and Shaking steps of VNS. The two next properties give sufficient conditions for the feasibility of random moves in 3–opt and double-bridge neighborhoods.

The size of the 3–opt neighborhood is $O(n^3)$. Once three arbitrary edges from the current tour are deleted, there are 8 possible ways to add three new edges to obtain a new tour. One among them, which does not change the orientation, is presented in Figure 1. We denote this special 3–opt by 3–opt*. Assume that we choose three non-consecutive customers from the tour $x_{i_1}, x_{i_3}, x_{i_5}$ and let $x_{i_2}, x_{i_4}, x_{i_6}$ be their successors on the tour, respectively. Without loss of generality, assume that the depot is located between the customers x_{i_6} and x_{i_1}.

Property 3.5 *If $L_{i_1} = L_{i_3} = L_{i_5}$ holds for the vehicle loads, then we can perform the 3–opt* modification without violating the feasibility condition.*

Proof. It is enough to show that the loads for the customers involved in the move remain the same. For example, the new loads for the vertices x_{i_4}, x_{i_2}, x_{i_6} remain the same,

$$L'_{i_4} = q_{x_1} + \ldots + q_{x_{i_1}} + q_{x_{i_4}} = L_{i_1} + q_{x_{i_4}} = L_{i_3} + q_{x_{i_4}} = L_{i_4}$$
$$L'_{i_2} = q_{x_1} + \ldots + q_{x_{i_1}} + q_{x_{i_4}} + \ldots + q_{x_{i_5}} + q_{x_{i_2}} = L_{i_5} + q_{x_{i_2}} = L_{i_1} + q_{x_{i_2}} = L_{i_2}$$
$$L'_{i_6} = q_{x_1} + \ldots + q_{x_{i_1}} + q_{x_{i_4}} + \ldots + q_{x_{i_5}} + q_{x_{i_2}} + \ldots + q_{x_{i_3}} + q_{x_{i_6}} = L_{i_3} + q_{x_{i_6}} = L_{i_5} + q_{x_{i_6}} = L_{i_6}.$$

□

For the double-bridge move, assume that we choose from the tour four non-consecutive customers x_{i_1}, x_{i_3}, x_{i_5}, x_{i_7} and let $x_{i_2}, x_{i_4}, x_{i_6}, x_{i_8}$ be their successors on the tour, respectively (see Figure 1).

Property 3.6 *If $L_{i_1} = L_{i_5}$ and $L_{i_3} = L_{i_7}$ then a new solution, obtained by the double bridge-move, is feasible.*

Proof. Without loss of generality, assume that the depot is located between customers x_{i_8} and x_{i_1}. If $L_{i_1} = L_{i_5}$ and $L_{i_3} = L_{i_7}$ holds for the vehicle loads, we can perform a double-bridge modification without violating the feasibility condition. For example, the new loads for the vertices x_{i_6}, x_{i_4}, x_{i_2}, x_{i_8} remain the same,

$$L'_{i_6} = q_{x_1} + \ldots + q_{x_{i_1}} + q_{x_{i_6}} = L_{i_1} + q_{x_{i_6}} = L_{i_5} + q_{x_{i_6}} = L_{i_6}$$
$$L'_{i_4} = q_{x_1} + \ldots + q_{x_{i_1}} + q_{x_{i_6}} + \ldots + q_{x_{i_7}} + q_{x_{i_4}} = L_{i_7} + q_{x_{i_4}} = L_{i_3} + q_{x_{i_4}} = L_{i_4}$$
$$L'_{i_2} = q_{x_1} + \ldots + q_{x_{i_1}} + q_{x_{i_6}} + \ldots + q_{x_{i_7}} + q_{x_{i_4}} + \ldots + q_{x_{i_5}} + q_{x_{i_2}} = L_{i_5} + q_{x_{i_2}} = L_{i_1} + q_{x_{i_2}} = L_{i_2}$$
$$L'_{i_8} = q_{x_1} + \ldots + q_{x_{i_1}} + q_{x_{i_6}} + \ldots + q_{x_{i_7}} + q_{x_{i_4}} + \ldots + q_{x_{i_5}} + q_{x_{i_2}} + \ldots q_{x_{i_3}} + q_{x_{i_8}} = L_{i_3} + q_{x_{i_8}} = L_{i_7} + q_{x_{i_8}} = L_{i_8}.$$

□

Therefore, we prune our shaking method by choosing four customers such that $L_{i_1} = L_{i_5}$ and $L_{i_3} = L_{i_7}$ and in this way feasibility is preserved.

3.4 Sequential and Mixed General VNS for 1-PDTSP

Finally we summarize briefly our two new heuristics for solving 1-PDTSP. Both are given in Algorithm 8. The first one uses sequential VND as a local search (see Algorithm 2) and we denote it by `Seq-GVNS`. The second one uses mixed nested VND as a local search and we denote it `Mix-GVNS`. Three neighborhoods are used in both sequential and mixed VND routines: 2–opt, forward insertion and backward insertion. For the 2–opt, we use of the tree data structures since it allows us a fast feasibility check in updating (see Properties 3.1 - 3.3 and the illustrative example in Figures 2 and 3). For insertion moves, we do not need to use tree data structure (see Property 3.4). In order to keep the feasibility after the perturbation of the incumbent solution, we design special kinds of random 3–opt and double-bridge moves (see Properties 3.5–3.6). The k^{th} shaking consists of k repetitions of random 3–opt* or double-bridge moves in such a way as to preserve feasibility (see Figure 1). A move is made if and only if a better solution is found.

For both GVNS methods, we use two usual parameters which should be estimated by the user: t_{max} – the total running time of the heuristic; $k_{max} = 2$ – the total number of neighborhoods used in the outer loop (the Shaking step). Note that ℓ_{max} (the total number of local search routines used in the VND) is equal to 3.

4 Computational Results

In this section the computational results of presented GVNS methods and their comparison with algorithms from the literature are given. All tests were carried out on the Intel Core 2 Duo T5800 2.0 GHz with 2 GB RAM, running the Linux operating system. The algorithms were coded in C++ programming language and compiled with `-O2` optimization. The memory used by our algorithm is as small as $O(n^2)$ and so it stays under 1 MB for even the largest test cases.

4.1 Test Instances

All experiments were tested on the benchmark instances of 1-PDTSP proposed in Hernández–Pérez et al. (2009). They are taken from `http://webpages.ull.es/users/hhperez/PDsite/#XM94`. These benchmark instances for $n \leq 500$ are randomly generated in the following way. The $n-1$ customers are randomly located in the square $[-500, 500] \times [-500, 500]$, each having a corresponding demand q_i randomly chosen from interval $[-10, 10]$. The depot is located in the origin $(0, 0)$ with demand $q_1 = -\sum_{i=2}^{n} q_i$. The travel cost $c(i, j)$ was computed as the Euclidean distance between points i and j. The vehicle capacities take values from the set $\{10, 20, 40\}$. For each n and Q, 10 random instances are generated. They are denoted with A, B, ..., I. In order to save space, we report the average values obtained on those 10 problem instances. However, the detailed results can be found in the Appendix and at our web page: `http://www.mi.sanu.ac.rs/~nenad/pdtsp`.

All the instances may be divided into two classes: small instances with a value of n from $\{20, 30, 40, 50, 60\}$ and large instances with n from $\{100, 200, 300, 400, 500\}$. Note that, for a given number of customers n, the smaller Q, the more difficult problem, is obtained. However, if Q is too large (let us say $Q = 1000$) the solution of the 1-PDTSP coincides with the solution of the TSP. Similarly to the above, we constructed instances with $n = 1000$ customers. Thus, we tested our code with instances which were twice as larger as the largest of those used in the past.

The optimal solutions for all small instances are known since they are obtained by means of the branch-and-cut algorithm in Hernández–Pérez and Salazar-González (2004b). For large instances, optimal solutions are not known. However, the best known results can be found in Hernández–Pérez et al. (2009) and Zhao et al. (2009a).

4.2 Preliminary Experimentation

Initial solutions. The first tests that were conducted are multistart initializations with or without local searches, 2–opt and forward/backward insertions. In the initialization, we generated $ms \in \{1, 10, 50, 100\}$) times the initial tour: if a feasible solution is found, we use it as the starting tour, otherwise we use the solution with minimal infeasibility (defined as $\max_{i \in V} L_i(x) - \min_{i \in V} L_i(x)$). In the first case we perform local search procedures in order to improve the initial tour, while in the second case we do not perform further modifications of the tour. Table 1 contains the results of this experiment: in the first row the number of feasible tour lengths found in 1000 starts is shown; in the second, third and fourth rows the best, worst and average feasible tour lengths are shown; in the fifth, sixth and seventh rows the best, worst and average tour lengths are shown (including infeasible tours); in the eighth row the average execution time is shown. According to these results, we conclude that for a large enough ms after the first local search procedure we have a feasible solution in almost all restarts. This is of great importance for the rest of the algorithm.

Table 1: Multistart comparison for $n = 400$ on A and I instances with local search

	$Q = 10$				$Q = 20$			
	$ms = 1$	$ms = 10$	$ms = 50$	$ms = 100$	$ms = 1$	$ms = 10$	$ms = 50$	$ms = 100$
Num. feasible	665	797	955	990	1000	1000	1000	1000
Min feasible	33106.50	33167.60	32988.20	32656.90	22665.80	22665.80	22665.80	22665.80
Max feasible	36555.90	35667.10	37402.00	37402.00	26739.70	26739.70	26739.70	26739.70
Avg. feasible	34560.79	34779.50	34912.70	34966.05	24389.75	24336.90	24336.90	24336.90
Min of all	33106.50	33167.60	32988.20	32656.90	22665.80	22665.80	22665.80	22665.80
Max of all	93025.00	93750.90	92266.00	91599.70	26739.70	26739.70	26739.70	26739.70
Avg. of all	46955.69	44316.48	37227.58	35494.67	24389.75	24336.90	24336.90	24336.90
Avg. time	0.55	0.61	0.74	0.72	0.36	0.36	0.38	0.36
Num. feasible	400	427	562	651	1000	1000	1000	1000
Min feasible	30760.80	30760.80	31020.40	30605.80	21779.90	21780.70	21780.70	21780.70
Max feasible	83924.60	83924.60	86255.90	88312.70	26537.10	25634.40	25634.40	25634.40
Avg. feasible	32946.10	32708.91	32919.97	33023.48	23343.05	23306.95	23306.95	23306.95
Min of all	30760.80	30760.80	31020.40	30605.80	21779.90	21780.70	21780.70	21780.70
Max of all	92022.00	92601.00	92766.10	91611.30	26537.10	25634.40	25634.40	25634.40
Avg. of all	52513.70	58327.25	54465.37	50447.75	23343.05	23306.95	23306.95	23306.95
Avg. time	0.45	0.44	0.50	0.50	0.32	0.32	0.34	0.33

Multi-start local searches. In order to clearly see the impact of each neighborhood on the solution quality, we run 1000 times six local search heuristics: (a) forward insertion; (b) backward insertion; (c) 2–opt; (d) `Seq-VND`, (e) `Seq-VND3` (uses full 3–opt as the last in the VND list) and (f) `Mix-VND`. These heuristics are tested on instances with $n = 200$ and $n = 400$, in both $Q = 10$. The results are reported in Table 2, where columns 4, 5 and 6 give minimum, average and maximum % deviation from the best known solution, respectively. The last column reports the average computing time spent to reach local minima.

Table 2: Comparison of different local search algorithms on two instances

n	Q	Local Search	Min. % dev	Max. % dev	Avg. % dev	Avg. time
200	10	forward-insertion	95.650	255.517	181.195	0.088
		backward-insertion	94.753	252.827	186.318	0.081
		2–opt	13.882	242.910	32.433	0.138
		Seq–VND	12.275	242.910	27.808	0.163
		Seq–VND–3	8.991	242.910	24.309	0.478
		Mix–VND	1.269	242.910	12.958	2.989
400	10	forward-insertion	78.320	218.770	165.603	0.385
		backward-insertion	81.881	218.770	169.416	0.317
		2–opt	12.078	217.104	21.852	0.831
		Seq–VND	10.684	217.104	18.954	0.769
		Seq–VND–3	8.951	203.738	16.035	4.062
		Mix–VND	1.492	217.104	6.573	26.569

Regarding the solution quality of a single neighborhood structure, it appears that the 2–opt significantly outperforms both forward and backward insertion. Comparing 3 VND heuristics we see that `Mix-VND` takes much more time than other two, but it is still able to get a very good solution. It is also clear that including 3–opt in VND is not beneficial enough: for just 2% better average performance, more than 5 times as much computer time is spent. So we decided to keep `Seq-VND` and `Mix-VND` in our further experiments.

Results on instances with $n = 200$ are also illustrated on a distance-to-target diagram (Figure 4), where each local minimum is represented by the point (u_i, v_i), $i = 1, \ldots, 1000$. The best known solution is in origin: u_i is the distance of local minima i to the best known solution and v_i represents its % deviation from the best known objective function value. The distance between any two tours is calculated as the number of their different edges. The comparison between local searches are clearly illustrated in Figure 4. It is interesting to note that with all 6 neighborhoods local minima of bad quality are detected, even with `Mix-VND`.

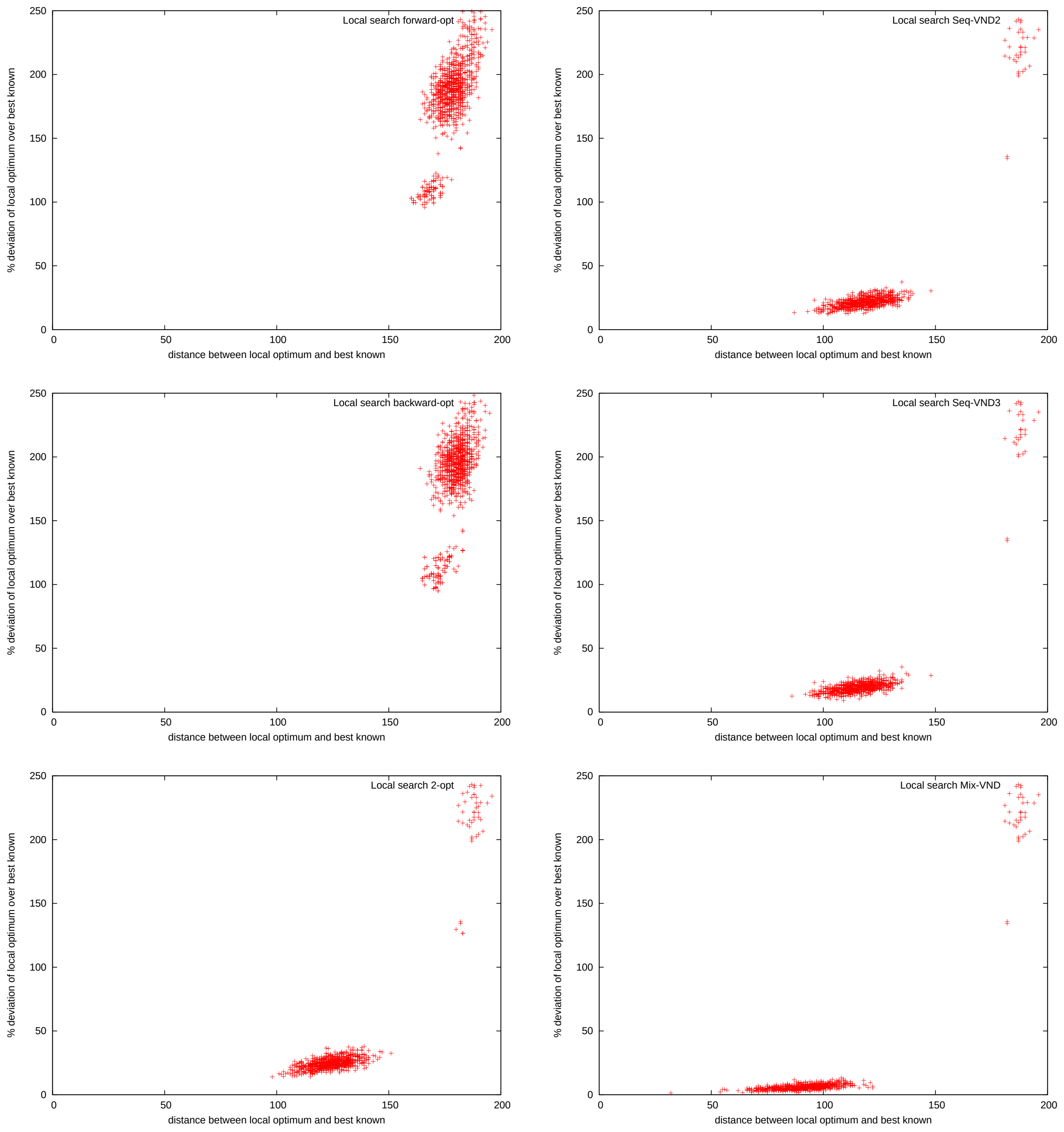

Figure 4: Distribution of 1000 local minima on distance-to-target diagram for different local search algorithms

VNS without 3–opt versus VNS with 3–opt. As mentioned earlier, we also implemented a 3–opt neighborhood with binary index tree structure. It is considered as the last neighborhood in the sequential VND. Let us denote it with VND-3 and with VND-2 the previous VND. In Table 3 are presented the results obtained in 20 runs of VNS with VND-2 and VNS with VND-3 as local search procedures. From this table we can conclude that the results obtained with 3–opt and the results obtained without 3–opt are very similar (in both solution quality and execution time). Based on these findings and with the wish to use a more user-friendly heuristic, we choose VNS with VND-2 as a local search for our final heuristics.

Table 3: Comparison of VND without 3–opt (VND-2) and with 3–opt (VND-3)

Parameters		VNS (with VND-2)			VNS (with VND-3)		
n	Q	Best	Average	Time	Best	Average	Time
200	10	18699.1	18989.62	49.74	18709.1	19000.88	51.70
200	20	13385.1	13627.38	37.60	13391.4	13637.40	40.10
200	40	11223.8	11323.93	18.00	11236.4	11338.02	19.73
400	10	25545.1	25962.14	165.95	25555.6	25974.50	167.58
400	20	18518.9	18786.59	69.41	18530.7	18801.92	71.05
400	40	15680.0	15803.19	48.67	15687.2	15821.80	50.62

4.3 Main Computational Results

The results of VNS for small and large instances are presented in Table 4 and Table 5 respectively. For each $Q \in \{10, 20, 40\}$, we report the best and the average lengths of the vehicle tour and the execution time. For small instances with $n \leq 60$, we report the optimal length of the vehicle tours.

The optimal solutions for all small instances are known, since they are obtained by means of the branch-and-cut algorithm in Hernández–Pérez and Salazar-González (2004b). For large instances, optimal solutions are not known. However, the best known results can be found in Hernández–Pérez et al. (2009) and Zhao et al. (2009a). Note that the CPU times for GA are missing in Table 4 and Table 5, since Zhao et al. (2009a) report only the average running times for large instances on $n = 100, 200, 300, 400, 500$ nodes and $Q = 10$ with the conclusion that their GA method is faster than the hybrid heuristic algorithm of Hernández–Pérez et al. (2009).

Table 4: Comparison on small benchmark instances

Parameters		Best	Seq-GVNS		Mix-GVNS		GRASP VND		GA		CPU time			
n	Q	known	Best	Average	Best	Average	Best	Average	Best	Average	Seq-GVNS	Mix-GVNS	GRASP	GA
20	10	5402.50	0.00	0.38	0.02	0.68	0.00	0.00	0.00	0.00	0.03	0.47	0.07	
	20	4237.30	0.00	0.10	0.00	0.10	0.00	0.00			0.01	0.10	0.03	
	40	3977.40	0.00	0.37	0.00	0.28	0.00	0.00			0.01	0.04	0.02	
30	10	6576.10	0.00	0.03	0.00	0.00	0.00	0.01	0.00	0.00	0.14	0.70	0.33	
	20	5017.20	0.00	0.00	0.00	0.00	0.00	0.00			0.04	0.11	0.08	
	40	4604.50	0.00	0.00	0.00	0.00	0.00	0.00			0.01	0.04	0.05	
40	10	7189.30	0.00	0.07	0.00	0.00	0.00	0.40	0.00	0.04	0.78	1.95	0.80	
	20	5500.10	0.00	0.00	0.00	0.00	0.00	0.00			0.10	0.15	0.15	
	40	5142.50	0.00	0.00	0.00	0.00	0.00	0.00			0.03	0.18	0.09	
50	10	8681.70	0.00	0.58	0.00	0.06	0.01	1.14	0.00	0.39	1.37	4.06	1.51	
	20	6537.70	0.00	0.09	0.00	0.00	0.00	0.16			0.48	1.29	0.34	
	40	5945.90	0.00	0.01	0.00	0.00	0.00	0.00			0.14	0.83	0.17	
60	10	9165.80	0.00	0.70	0.00	0.11	0.15	2.03	0.00	0.74	2.22	7.02	2.40	
	20	6956.00	0.03	0.12	0.00	0.00	0.08	0.34			0.73	2.81	0.49	
	40	6361.70	0.00	0.00	0.00	0.00	0.00	0.00			0.26	0.88	0.26	

Table 5: Comparison on large benchmark instances

Parameters		Best	Seq-GVNS		Mix-GVNS		GRASP VND		GA		CPU time			
n	Q	known	Best	Average	Best	Average	Best	Average	Best	Average	Seq-GVNS	Mix-GVNS	GRASP	GA
100	10	12718.60	0.29	1.52	0.0	0.35	1.88	4.62	0.96	1.85	9.96	23.79	8.85	
	20	9357.60	0.02	0.96	0.00	0.18	0.65	2.55			5.89	14.95	2.22	
	40	8165.40	0.00	0.20	0.00	0.04	0.00	0.57			1.84	5.97	0.69	
200	10	18578.80	0.65	2.21	0.00	1.05	4.81	7.50	2.70	4.09	49.74	75.69	41.77	
	20	13319.00	0.50	2.32	0.00	0.91	4.58	6.86			37.60	67.58	17.37	
	40	11214.80	0.08	0.97	0.00	0.31	1.34	3.18			18.00	42.32	4.35	
300	10	22935.30	0.83	2.39	0.00	1.47	5.30	7.67	4.20	5.62	104.61	122.83	117.86	
	20	16313.40	0.88	2.60	0.00	1.43	6.60	8.62			38.74	115.93	50.90	
	40	13671.40	0.41	1.40	0.00	0.56	2.85	4.80			24.91	88.63	12.89	
400	10	25467.20	0.31	1.94	0.00	1.12	5.66	7.68	4.02	5.82	165.95	165.44	220.40	
	20	18407.00	0.61	2.06	0.00	1.30	6.49	8.61			69.41	152.36	91.73	
	40	15602.90	0.49	1.28	0.00	0.70	3.41	5.20			48.67	144.29	23.92	
500	10	28774.20	0.00	1.54	0.10	1.28	5.80	7.76	5.57	7.37	124.14	209.76	391.01	
	20	20927.00	0.17	1.70	0.00	1.38	6.53	8.43			107.01	194.76	164.77	
	40	17495.50	0.41	1.50	0.00	0.86	4.47	6.10			89.81	193.52	43.98	
1000	10	44744.20	0.96	19.74	0.00	1.52					349.59	393.22		
	20	31661.10	1.64	3.57	0.00	1.56	7.69	8.95			478.08	441.03	618.33	
	40	25450.00	1.20	2.66	0.00	1.28	6.64	8.14			474.72	430.16	440.00	

In this study, we also compare our VNS with

- heuristics from Hernández–Pérez and Salazar-González (2004a) (only the best solution found);
- the hybrid GRASP/VND metaheuristic algorithm from Hernández–Pérez et al. (2009) (the best solution, average solution and execution time);
- the genetic algorithm from Zhao et al. (2009a) (best solution, average solution and standard deviation).

Clearly, our approach improves the best known solutions in **all** large benchmark instances, as showed in Table 6, Table 7 and Table 8. Furthermore, in Table 9 we compare the hybrid GRASP/VND metaheuristic with our basic VNS algorithm. For $Q = 10$ the hybrid GRASP/VND metaheuristic has failed to find a feasible solution, while in all other cases our VNS finds better solutions (5 percent on average).

Based on these results, we conclude that:

(i) the VNS based heuristic improves the best known solutions in all benchmark instances.

(ii) VNS is robust. For some problem sizes, the worst average solution (out of 20) obtained by VNS is of better quality than the best ones obtained by other two methods. For example, compare the values of 19515.5, 19472.1, 19080.3 and 18578.8 from Table 6 in line with the best values obtained for $n = 200$ and $Q = 10$.

(iii) GA performs better than GRASP/VND, but instances with larger Q are not tested;

(iv) using our VNS approach, we are able to efficiently solve instances with $n = 1000$ customers.

5 Conclusion

We develop a General variable neighborhood search (GVNS) based heuristics for solving the One-commodity pickup-and-delivery travelling talesman problem (1-PDTSP). This problem contains two NP-hard problems: TSP and the feasibility checking of the tour.

GVNS is a variant of VNS which consists of the usual Shaking operator, but several neighborhoods are used in deterministic fashion in the local search step. For performing such a local search, known as the Variable Neighborhood Descent (VND), we propose the use of three different operators: 2–opt, insertion (forward and backward) and 3–opt. Moreover, we use those structures in two ways: sequentially (following the prescribed order) and nested. In order to store and update the incumbent solution, we implemented a binary index tree structure and proved several properties regarding the complexity of the feasibility checking step.

The Shaking operator which we designed uses 3–opt and double–bridge moves, maintaining the feasibility of the incumbent solution. Based on computational results, our method significantly improves the best-known solutions for all large instances in the literature. The average improvement over the previous state-of-the-art heuristics is up to 7%! For new large instances with $n = 1000$, the improvements reported are even larger. These results are obtained in less computing time than other methods, while keeping memory using reasonable.

These results are very encouraging and we are considering as future research topics the application of a similar VNS methodology to related pickup-and-delivery problems.

6 Appendix – Detailed Results for Large Instances

Table 6: Comparison of the results on large instances

Parameters	**UB2**	**GRASP/VND**			**GA**			**Seq-GVNS**			**Mix-GVNS**		
Name	Best	H-Best	H-Average	Time	Best	Average	Stdev	Best	Average	Time	Best	Average	Time
n100q10A	12042	11874	12087.60	8.48	11828	11922.60	71.30	**11669**	11850.35	9.75	**11669**	11702.45	25.90
n100q10B	13172	13288	13582.60	10.23	13114	13301.60	157.10	13005	13191.45	10.58	**12938**	12983.65	25.32
n100q10C	14063	14069	14421.30	10.27	13977	14095.20	147.20	13893	13999.90	12.79	**13893**	13900.55	23.44
n100q10D	14490	14542	14787.50	8.95	14253	14406.40	111.90	14386	14542.80	11.48	**14245**	14298.90	25.45
n100q10E	11546	11650	12502.60	6.13	11411	11436.40	52.40	**11403**	11668.15	7.50	**11403**	11422.40	19.71
n100q10F	12021	11734	12010.70	7.67	11644	11699.00	34.50	11626	11745.45	7.28	**11609**	11621.35	19.75
n100q10G	12170	12049	12366.90	7.82	12038	12120.20	104.80	**11866**	11986.95	9.19	**11866**	11912.45	22.36
n100q10H	13056	12892	13169.20	9.39	12818	12906.20	125.10	12656	12828.70	10.06	**12647**	12699.20	30.39
n100q10I	14191	14048	14390.20	7.94	14032	14137.20	95.90	13888	13985.55	8.70	**13751**	13863.40	20.35
n100q10J	13439	13430	13737.60	11.65	13297	13516.80	216.40	**13165**	13323.05	12.28	**13165**	13227.75	25.21
Average	13019.0	12957.6	13305.60	8.85	12841.2	12954.20	111.70	12755.7	12912.24	9.96	**12718.6**	12763.21	23.79
n200q10A	18013	18145	18564.00	36.00	17686	17987.00	201.90	17422	17687.75	42.35	**17029**	17336.70	73.08
n200q10B	18154	18520	18932.50	33.68	17798	18069.40	243.10	17546	17969.55	40.78	**17439**	17518.15	79.51
n200q10C	17305	16969	17280.30	41.01	16466	16751.20	245.80	16224	16424.60	38.76	**16128**	16301.15	71.86
n200q10D	21565	21848	22285.70	33.51	21306	21564.40	207.30	20878	21133.40	62.56	**20819**	21027.70	81.70
n200q10E	20033	19913	20643.20	39.75	19299	19713.00	358.90	18992	19331.35	43.20	**18854**	19095.10	70.31
n200q10F	22090	21949	22284.60	80.93	21910	22144.00	247.70	21303	21520.60	64.31	**21250**	21418.40	75.07
n200q10G	17956	18035	18627.70	28.58	17712	17797.80	80.60	17147	17448.85	44.92	**17056**	17249.05	71.59
n200q10H	21995	21463	22084.90	47.45	21276	21584.00	278.40	20777	21065.60	62.35	**20719**	20965.80	73.20
n200q10I	18695	18606	19184.80	34.31	18380	18509.80	149.60	18078	18282.65	47.55	**17846**	18004.25	80.94
n200q10J	19349	19273	19839.50	42.43	18970	19274.20	205.50	**18624**	19031.85	50.57	18648	18825.95	79.66
Average	19515.5	19472.1	19972.70	41.77	19080.3	19339.50	221.90	18699.1	18989.62	49.74	**18578.8**	18774.23	75.69
n300q10A	23244	23566	24052.90	112.51	23242	23592.00	265.10	22501	22831.90	111.33	**22134**	22611.70	133.35
n300q10B	23256	23187	23845.60	109.55	22934	23028.60	114.90	22264	22653.80	126.49	**22172**	22406.40	118.91
n300q10C	22276	21800	22516.60	104.58	21922	22083.40	189.60	21191	21468.35	124.20	**20919**	21191.75	118.98
n300q10D	26434	25971	26462.10	162.95	25883	26289.80	253.50	24902	25349.35	130.20	**24719**	25099.80	117.82
n300q10E	27931	27420	27892.10	139.56	27367	27923.80	358.50	26224	26688.55	137.44	**26061**	26612.35	117.99
n300q10F	25096	24852	25278.20	153.93	24826	25055.40	171.80	**23730**	24144.80	125.91	23739	23999.30	122.08
n300q10G	24363	24308	24760.50	151.22	23868	24300.60	412.00	23316	23655.40	133.34	**23210**	23492.75	118.95
n300q10H	22869	22684	23116.50	67.49	21625	21965.00	278.50	21414	21715.30	53.08	**21183**	21420.55	128.27
n300q10I	25157	24633	25492.60	76.72	24513	24959.20	330.10	23726	24099.70	58.08	**23503**	23815.40	119.92
n300q10J	23468	23086	23530.20	100.05	22810	23045.00	351.10	21989	22224.30	46.05	**21713**	22071.15	132.03
Average	24409.4	24150.7	24694.70	117.86	23899.0	24224.30	272.50	23125.7	23483.15	104.61	**22935.3**	23272.12	122.83
n400q10A	31821	31486	31912.00	282.00	31678	31964.40	309.90	30061	30505.75	179.45	**29904**	30274.30	174.08
n400q10B	24883	25243	25606.40	204.21	24262	24752.40	283.20	23744	24279.15	175.10	**23658**	23917.50	162.85
n400q10C	29044	28942	29463.20	246.29	28741	29287.40	603.60	27782	28123.15	186.32	**27550**	27878.65	159.57
n400q10D	24639	24597	25308.60	142.84	24508	24794.80	320.10	23212	23574.45	182.68	**23194**	23426.05	160.61
n400q10E	25548	25644	26120.00	219.87	25071	25473.00	276.40	**24217**	24665.05	180.27	24258	24533.85	159.67
n400q10F	27215	27169	27755.10	273.01	26681	27362.80	411.60	26197	26475.15	186.58	**25900**	26243.85	174.16
n400q10G	24728	24626	25088.40	181.55	23891	24290.40	273.00	**23189**	23589.70	178.19	23366	23471.55	155.69
n400q10H	26191	26030	26468.80	220.74	25348	25811.40	351.50	24550	25002.60	185.46	**24392**	24703.35	161.33
n400q10I	28992	29154	29596.60	202.43	28714	29261.60	488.70	27730	28129.20	124.04	**27640**	27980.75	174.87
n400q10J	26607	26204	26916.20	231.03	26010	26489.40	281.60	**24769**	25277.20	81.36	24810	25096.40	171.56
Average	26966.8	26909.5	27423.50	220.40	26490.4	26948.80	360.00	25545.1	25962.14	165.95	**25467.2**	25752.63	165.44
n500q10A	29536	28742	29323.60	400.63	28857	29258.80	478.30	27301	27711.85	110.99	**27159**	27557.05	222.29
n500q10B	27370	27335	27711.10	332.67	26648	27454.80	525.70	25728	26021.15	102.19	**25610**	25959.85	213.21
n500q10C	31494	31108	31692.70	440.35	30701	31426.80	609.40	**29165**	29768.90	134.43	29365	29778.50	213.69
n500q10D	31752	30794	31428.40	426.51	30994	31442.20	376.90	**29264**	29650.30	127.22	29346	29639.40	184.32
n500q10E	31555	30674	31371.70	398.15	30905	31154.60	231.30	**29155**	29527.40	139.40	29174	29571.95	193.95
n500q10F	28957	29258	29812.30	263.14	28882	29241.00	244.90	27581	27873.90	106.17	**27513**	27809.65	215.39
n500q10G	27492	27198	27958.20	306.38	27107	27473.00	212.50	25765	26065.70	107.66	**25741**	25981.50	212.66
n500q10H	37185	36857	37361.10	600.00	37626	38142.40	258.80	**34995**	35587.55	160.90	35241	35562.60	212.82
n500q10I	31612	31045	31536.00	316.74	30796	31044.60	306.00	**29272**	29910.45	124.94	29291	29662.80	210.89
n500q10J	31412	31423	31877.90	425.56	31255	32310.00	617.90	**29516**	30042.40	127.46	29586	29910.10	218.42
Average	30836.5	30443.4	31007.30	391.01	30377.1	30894.80	386.20	**28774.2**	29215.96	124.14	28802.6	29143.34	209.76

Table 7: Comparison of the results on large instances for $Q = 20$

Parameters		**UB2**	**GRASP/VND**			**Seq-GVNS**			**Mix-GVNS**		
Name	n	Best	H-Best	H-Average	Time	Best	Average	Time	Best	Average	Time
n100q20A	100	8768	8616	8779.20	1.63	**8613**	8645.55	5.26	**8613**	8620.70	5.79
n100q20B	100	9629	9536	9686.90	2.51	**9469**	9554.20	3.89	**9469**	9524.60	4.90
n100q20C	100	10099	9993	10191.20	3.03	**9928**	9995.70	5.17	**9928**	9951.35	18.87
n100q20D	100	10464	10064	10340.70	3.07	**10015**	10133.65	7.63	**10015**	10015.00	10.92
n100q20E	100	8929	8838	8986.20	1.42	**8829**	8914.00	4.04	**8829**	8833.85	17.66
n100q20F	100	9056	9029	9106.20	1.63	8975	9019.40	4.71	**8959**	8977.30	20.63
n100q20G	100	9022	8865	9078.50	1.58	**8838**	8932.60	7.08	**8838**	8838.00	15.33
n100q20H	100	9708	9495	9681.00	2.36	9413	9525.25	6.36	**9406**	9431.55	17.25
n100q20I	100	10144	10005	10192.70	2.41	**9868**	9986.15	8.04	**9868**	9877.05	12.85
n100q20J	100	9835	9742	9922.70	2.56	**9651**	9769.45	6.72	**9651**	9673.45	25.31
Average	100	9565.4	9418.3	9596.53	2.22	9359.9	9447.60	5.89	**9357.6**	9374.29	14.95
n200q20A	200	13455	13422	13714.80	11.01	12860	13194.70	27.95	**12824**	12995.55	65.81
n200q20B	200	13242	13419	13714.80	12.32	12704	13059.10	28.77	**12689**	12877.90	62.35
n200q20C	200	12264	12314	12678.50	8.93	12047	12152.65	26.80	**11999**	12008.60	47.11
n200q20D	200	15387	15212	15548.60	24.59	14641	14876.65	42.27	**14605**	14707.75	68.86
n200q20E	200	14109	14066	14298.30	18.41	13469	13660.60	42.79	**13370**	13481.95	71.60
n200q20F	200	15105	15167	15542.00	30.87	14548	14805.05	56.40	**14490**	14612.60	75.37
n200q20G	200	13203	13200	13495.70	11.43	12713	12959.75	34.26	**12689**	12785.70	61.14
n200q20H	200	15518	15278	15571.80	26.74	14606	14882.90	51.46	**14543**	14715.00	80.64
n200q20I	200	13082	13338	13597.80	13.63	12860	13041.30	27.63	**12690**	12810.75	68.83
n200q20J	200	14043	13870	14159.40	15.80	13403	13641.05	37.69	**13291**	13402.80	74.12
Average	200	13940.8	13928.6	14232.17	17.37	13385.1	13627.38	37.60	**13319.0**	13439.86	67.58
n300q20A	300	16830	16920	17242.80	41.74	15982	16197.00	36.43	**15868**	16031.35	118.15
n300q20B	300	16844	17050	17248.40	39.31	15977	16363.70	35.73	**15932**	16157.05	106.65
n300q20C	300	16548	16364	16661.10	34.11	15502	15740.65	34.70	**15417**	15556.90	103.18
n300q20D	300	18024	18178	18651.70	61.75	17299	17533.35	44.25	**17105**	17354.00	119.37
n300q20E	300	19130	18715	19088.50	86.47	**17747**	18033.65	44.03	17829	17954.05	115.69
n300q20F	300	18216	18126	18387.10	63.61	17015	17294.55	40.97	**16737**	17035.15	129.37
n300q20G	300	17490	17363	17759.40	53.70	16548	16879.15	42.85	**16433**	16653.75	123.59
n300q20H	300	16759	16725	16997.70	32.67	15733	16096.55	35.02	**15591**	15933.80	115.97
n300q20I	300	18048	17654	17996.00	60.08	16740	17024.60	40.97	**16449**	16822.20	110.55
n300q20J	300	17027	16811	17168.50	35.51	16019	16208.60	32.42	**15773**	15971.60	116.82
Average	300	17491.6	17390.6	17720.12	50.90	16456.2	16737.18	38.74	**16313.4**	16546.99	115.93
n400q20A	400	21741	21617	22042.20	198.75	20574	20837.15	82.67	**20344**	20682.05	145.60
n400q20B	400	18459	19021	19260.70	60.09	17979	18251.95	63.51	**17883**	18029.60	154.34
n400q20C	400	20827	20765	21172.00	125.79	19433	19799.80	83.54	**19418**	19653.20	151.05
n400q20D	400	18443	18375	18767.00	49.26	17267	17480.25	63.78	**17133**	17393.70	148.73
n400q20E	400	18598	18764	19153.60	64.63	17659	18002.95	59.56	**17481**	17772.85	149.91
n400q20F	400	20112	19941	20223.80	85.24	18796	19030.70	67.91	**18549**	18831.20	140.42
n400q20G	400	18695	18624	18900.50	55.50	**17512**	17775.55	60.58	17515	17707.50	154.44
n400q20H	400	18882	18829	19468.30	73.13	18039	18261.85	69.45	**17989**	18199.70	162.23
n400q20I	400	20682	20610	21120.30	132.23	**19561**	19825.90	75.78	19563	19756.95	148.60
n400q20J	400	18958	19478	19804.60	72.65	18369	18599.80	67.27	**18195**	18444.55	168.29
Average	400	19539.7	19602.4	19991.30	91.73	18518.9	18786.59	69.41	**18407.0**	18647.13	152.36
n500q20A	500	21702	21585	21758.20	121.52	**20057**	20382.10	107.76	20092	20292.00	186.75
n500q20B	500	20523	20762	21082.20	92.81	19424	19669.30	97.40	**19333**	19586.30	187.88
n500q20C	500	23034	22738	23108.70	177.94	**21100**	21583.05	109.96	21169	21480.60	185.50
n500q20D	500	22774	22737	23032.20	163.36	**21111**	21554.40	104.82	21141	21442.45	201.96
n500q20E	500	22775	22480	22812.10	192.74	21213	21476.65	107.78	**21183**	21435.85	197.43
n500q20F	500	21745	21679	22022.80	121.52	20494	20749.10	93.72	**20334**	20663.00	204.66
n500q20G	500	20325	20617	20983.30	93.22	**19463**	19715.75	98.66	19501	19733.80	214.66
n500q20H	500	26250	25383	25968.70	347.42	23993	24279.00	128.46	**23822**	24246.85	188.82
n500q20I	500	22472	22442	23083.60	172.67	**21386**	21722.05	111.07	21392	21644.35	186.14
n500q20J	500	22756	22517	23054.50	164.47	21388	21697.10	110.43	**21303**	21641.80	193.77
Average	500	22435.6	22294.0	22690.63	164.77	20962.9	21282.85	107.01	**20927.0**	21216.70	194.76

Table 8: Comparison of the results on large instances for $Q = 40$

Parameters		**UB2**	**GRASP/VND**			**Seq-GVNS**			**Mix-GVNS**		
Name	n	Best	H-Best	H-Average	Time	Best	Average	Time	Best	Average	Time
n100q40A	100	7938	**7938**	7941.80	0.60	**7938**	7938.00	0.36	**7938**	7938.00	0.91
n100q40B	100	8144	**8124**	8182.60	0.72	**8124**	8151.90	2.53	**8124**	8127.00	7.71
n100q40C	100	8441	**8441**	8514.50	0.81	**8441**	8464.70	1.08	**8441**	8441.00	2.15
n100q40D	100	8380	**8264**	8360.00	0.82	**8264**	8330.20	3.92	**8264**	8293.90	17.89
n100q40E	100	7960	**7960**	7996.50	0.58	**7960**	7960.00	0.74	**7960**	7960.00	1.75
n100q40F	100	8074	**8074**	8116.10	0.56	**8074**	8080.10	1.65	**8074**	8074.00	2.40
n100q40G	100	8183	**8168**	8189.00	0.60	**8168**	8175.70	1.30	**8168**	8168.65	9.34
n100q40H	100	7992	**7992**	8022.30	0.74	**7992**	7992.00	0.87	**7992**	7992.00	0.54
n100q40I	100	8484	8440	8504.10	0.71	**8438**	8462.90	3.04	**8438**	8439.75	11.99
n100q40J	100	8255	**8255**	8289.10	0.71	**8255**	8257.85	2.86	**8255**	8255.00	5.04
Average	100	8185.1	8165.6	8211.60	0.69	**8165.4**	8181.34	1.84	**8165.4**	8168.93	5.97
n200q40A	200	11136	11156	11369.30	3.42	**11039**	11172.75	18.78	**11039**	11055.65	44.68
n200q40B	200	11305	11296	11489.60	3.48	11207	11274.00	15.65	**11178**	11213.45	54.60
n200q40C	200	10919	10849	11038.40	3.01	**10833**	10860.30	12.21	**10833**	10833.00	9.86
n200q40D	200	12002	11802	12037.40	5.39	**11533**	11683.60	23.79	**11533**	11598.40	45.26
n200q40E	200	11276	11237	11474.70	4.70	**11042**	11219.90	19.56	**11042**	11108.95	54.15
n200q40F	200	11931	11836	11988.10	6.35	11530	11684.05	21.96	**11528**	11577.50	56.09
n200q40G	200	11174	11154	11302.00	3.54	11008	11093.35	16.25	**10987**	11029.85	46.86
n200q40H	200	12234	12088	12430.20	5.87	11990	12067.60	30.91	**11982**	12013.00	35.81
n200q40I	200	11272	11115	11298.10	3.66	**11040**	11100.15	10.70	**11040**	11045.40	29.87
n200q40J	200	11181	11123	11281.50	4.07	11016	11083.60	10.23	**10986**	11016.95	45.99
Average	200	11443.0	11365.6	11570.93	4.35	11223.8	11323.93	18.00	**11214.8**	11249.22	42.32
n300q40A	300	13670	13787	14008.10	11.32	13517	13619.30	27.58	**13429**	13486.75	100.28
n300q40B	300	13881	13875	14127.20	10.95	13571	13723.05	22.19	**13563**	13615.70	81.63
n300q40C	300	13489	13642	13792.70	9.79	**13263**	13373.10	14.51	**13263**	13310.00	71.69
n300q40D	300	14477	14426	14733.60	14.66	**14008**	14177.95	26.10	14016	14084.50	100.80
n300q40E	300	14616	14521	14902.10	20.21	14174	14338.65	34.38	**14098**	14240.95	96.56
n300q40F	300	14390	14345	14665.70	15.30	13998	14165.25	27.01	**13900**	14005.20	89.44
n300q40G	300	14299	14151	14382.60	13.10	13806	13939.10	22.55	**13730**	13826.40	98.94
n300q40H	300	13816	13674	14047.10	9.39	13505	13623.20	23.57	**13463**	13513.75	66.06
n300q40I	300	14396	14232	14489.20	14.58	13920	14023.60	29.15	**13768**	13874.40	92.57
n300q40J	300	13759	13963	14127.70	9.58	13514	13641.85	22.03	**13484**	13518.05	88.29
Average	300	14079.3	14061.6	14327.60	12.89	13727.6	13862.51	24.91	**13671.4**	13747.57	88.63
n400q40A	400	16966	16939	17198.20	41.83	**16259**	16465.30	60.47	16279	16371.10	144.34
n400q40B	400	16027	16013	16217.00	18.56	15554	15692.45	45.38	**15448**	15544.95	125.85
n400q40C	400	16506	16588	16964.10	31.03	16107	16242.00	58.88	**16045**	16169.20	143.26
n400q40D	400	15691	15801	16033.20	15.39	15526	15573.75	37.86	**15429**	15504.05	137.77
n400q40E	400	15658	15638	15906.50	18.91	15215	15314.90	42.51	**15133**	15235.10	146.57
n400q40F	400	16085	16373	16541.00	23.52	15816	15910.95	44.44	**15724**	15802.85	138.35
n400q40G	400	15603	15716	15955.20	16.92	15362	15434.15	47.00	**15264**	15405.40	140.09
n400q40H	400	15936	15848	16236.60	20.01	15469	15674.30	46.27	**15404**	15574.20	164.97
n400q40I	400	16554	16477	16809.60	32.49	15930	16042.10	54.90	**15825**	15947.65	154.74
n400q40J	400	15678	15951	16286.30	20.52	15562	15682.05	48.97	**15478**	15564.55	146.96
Average	400	16070.4	16134.4	16414.77	23.92	15680.0	15803.20	48.67	**15602.9**	15711.91	144.29
n500q40A	500	17966	17840	18064.60	36.50	17190	17323.45	78.01	**17101**	17237.05	216.17
n500q40B	500	17161	17574	17898.50	28.87	17107	17235.20	71.94	**17056**	17149.10	216.25
n500q40C	500	18529	18498	18758.40	45.04	17752	17979.20	102.32	**17675**	17832.60	190.89
n500q40D	500	18307	18573	18838.40	44.01	17838	17987.00	87.24	**17791**	17972.55	200.26
n500q40E	500	18351	18335	18608.80	47.78	17598	17797.80	87.45	**17539**	17683.10	181.44
n500q40F	500	18101	17976	18263.60	36.48	**17249**	17473.45	83.71	17284	17382.25	194.50
n500q40G	500	17697	17600	17894.20	29.68	17045	17209.65	77.65	**16936**	17098.40	184.58
n500q40H	500	19633	19619	19881.40	79.68	18645	18846.95	112.93	**18588**	18724.25	183.00
n500q40I	500	18349	18322	18618.40	47.21	17538	17802.05	93.14	**17398**	17604.60	173.18
n500q40J	500	18446	18445	18796.70	44.57	17706	17927.85	103.67	**17587**	17782.10	194.93
Average	500	18254.0	18278.2	18562.30	43.98	17566.8	17758.26	89.81	**17495.5**	17646.60	193.52

Table 9: Comparison of basic VNS and GRASP/VND for $n = 1000$

Name	**GRASP/VND**			**Seq-GVNS**			**Mix-GVNS**		
Name	Best	Average	Time	Best	Average	Time	Best	Average	Time
n1000q10A				45618	50922.40	380.91	**44846**	45320.30	408.93
n1000q10B				48857	71844.40	257.69	**47565**	48034.55	363.68
n1000q10C				46188	59191.45	322.52	**45397**	49355.10	419.24
n1000q10D				**42100**	54614.30	303.89	42109	48600.95	364.46
n1000q10E				43125	51783.00	266.54	**43083**	43679.65	315.05
n1000q10F				**45376**	56567.95	342.57	45531	46584.00	445.90
n1000q10G				42432	47030.35	339.12	**41879**	42962.25	353.93
n1000q10H				43352	44029.95	436.84	**43005**	43547.10	409.78
n1000q10I				43577	48055.95	401.83	**43449**	44009.05	436.77
n1000q10J				51108	51718.70	443.99	**50578**	51555.65	414.49
n1000q20A	33605.00	34088.90	630	32473	33010.05	481.08	**31967**	32612.3	458.12
n1000q20B	34762.00	35308.90	625	33738	34547.75	483.62	**33586**	34136.75	472.27
n1000q20C	33193.00	33724.75	620	31948	32825.95	485.02	**31862**	32471.3	471.98
n1000q20D	31846.00	32255.40	615	30610	30975.00	473.79	**30226**	30565.15	432.29
n1000q20E	32370.00	32751.25	615	31114	31563.20	483.46	**30642**	30995.95	412.8
n1000q20F	33528.00	33966.70	625	32204	33052.75	486.16	**31894**	32396.6	428.7
n1000q20G				30621	31279.10	479.76	**30337**	30701.5	434.46
n1000q20H	32637.00	32996.20	610	31183	31833.55	457.20	**30577**	31103.9	429.08
n1000q20I	33270.00	33599.15	615	32125	32600.95	474.75	**30944**	31521.25	424.14
n1000q20J	36385.00	36892.50	630	35774	36230.85	475.98	**34576**	35046.85	446.44
n1000q40A	27093	27402.80	430	25979	26481.45	467.87	**25652**	25952	379.83
n1000q40B	27099	27794.45	525	26503	26958.85	478.24	**26011**	26350.25	399.77
n1000q40C	26820	27075.00	450	25705	26011.30	477.04	**25295**	25683.8	430.57
n1000q40D	26137	26338.85	340	24797	25225.30	468.30	**24816**	25030.85	428.32
n1000q40E	26273	26701.95	335	25309	25560.70	478.95	**25003**	25348.3	414.67
n1000q40F	26820	27178.55	440	25780	26175.40	465.80	**25376**	25737.5	445.63
n1000q40G	26345	26606.35	310	25324	25472.30	477.62	**24696**	25149.8	447.23
n1000q40H	26600	26761.60	350	25328	25718.40	472.42	**25142**	25472.35	424.07
n1000q40I	26488	27083.45	360	25728	26070.05	482.05	**25574**	25795.15	469.01
n1000q40J	28335	28718.55	610	27103	27591.65	478.94	**26935**	27239.05	462.51

Bibliography

Anily S., Bramel J. 1999. Approximation Algorithms for the Capacitated Traveling Salesman Problem with Pickups and Deliveries. *Naval Research Logistics* **46** 654–670.

Brimberg J., P. Hansen, N. Mladenović. 2010. Attraction probabilities in Variable neighborhood search. *4OR* **8** 181–194.

Chalasani P., Motwani. 1999. Approximating capacitated routing and delivery problems. *SIAM Journal on Computing* **28** 2133–2149.

Cormen, T. H., Leiserson, C. E., Rivest, R. L., Stein, C., 2001. Introduction to Algorithms. Second Edition, MIT Press.

Fenwick, P. M. 1994. A new data structure for cumulative frequency tables. *Software – Practice and Experience* **24(3)** 327–336.

Fiechter, C.-N. 1994. A Parallel Tabu Search Algorithm for Large Traveling Salesman Problems. *Discrete Applied Math* **51** 243-267.

Gamboa, D., C. Rego, and F. Glover. 2006. Implementation analysis of efficient heuristic algorithms for the traveling salesman problem. *Computers & Operations Research* **33(4)** 1154–1172.

Gendreau, M., G. Laporte, D. Vigo. 1999. Heuristics for the traveling salesman problem with pickup and delivery. *Computers & Operations Research* **26** 699–714.

Gendreau, M., A. Hertz, G. Laporte. 1997. An approximation algorithm for the traveling salesman problem with backhauls. *Oper. Res.* **45** 639–641.

Guan, D. J. 1998. Routing a vehicle of capacity greater than one. *Discrete Appl. Math.* **81** 41–57.

Hansen, P., N. Mladenović. 2006. First vs. best improvement: an empirical study. *Discrete Applied Mathematics* **154** 802–817.

Hansen, P., N. Mladenović, J. A. M. Pérez, 2008. Variable neighborhood search. *European Journal of Operational Research* **191** 593–595.

Hansen, P., N. Mladenović, Moreno J. A. Pérez. 2010. Variable neighborhood search: algorithms and applications. Annals of Operations Research **175** 367–407.

Hernández–Pérez, H. 2004. Traveling salesman problems with pickups and deliveries. PhD Thesis, University of La Laguna, Spain.

Hernández–Pérez, H., J. J. Salazar-González. 2004. Heuristics for the one-commodity pickup-and-delivery traveling salesman problem. *Transportation Science* **38** 245–255.

Hernández–Pérez, H., I. Rodríguez-Martín, J. J. Salazar–González. 2009. A hybrid GRASP/VND heuristic for the one-commodity pickup-and-delivery traveling salesman problem. *Computers & Operations Research* **36** 1639–1645.

Hernández–Pérez, H., J. J. Salazar-González. 2004. A branch-and-cut algorithm for a traveling salesman problem with pickup and delivery. *Discrete Applied Mathematics* **145** 126–39.

Hernández–Pérez, H., J. J. Salazar-González. 2007. The one-commodity pickup and delivery traveling salesman problem: Inequalities and algorithms. *Networks* **50** 258–272.

Ilić A., D. Urosević, J. Brimberg, N. Mladenović. 2010. Variable neighborhood search for solving the uncapacitated single allocation p-hub median problem. *European Journal of Operational Research* **206** 289–300.

Johnson, D.S., and L.A. McGeoch. 1996. The Traveling Salesman Problem: A Case Study in Local Optimization. In E.H.L. Aarts, and J.K. Lenstra (eds.), Local Search in Combinatorial Optimization, New York: Wiley.

Johnson D. S., L. A.McGeoch. 1997. The traveling salesman problem: a case study in local optimization. In: Aarts E.J.L., and J.K. Lenstra (Eds.) Local search in combinatorial optimization. Chichester, UK: Wiley.

Laporte, G. 1992. The traveling salesman problem: An overview of exact and approximate algorithms. *European Journal of Operational Research* **59(2)** 231–247.

Lin, S., B. W. Kernighan. 1973. An effective heuristic algorithm for the traveling salesman problem. *Operations Research* **21** 498–516.

Mladenović, N., P. Hansen. 1997. Variable neighborhood search. *Computers & Operations Research* **24** 1097–1100.

Mosheiov, G. 1994. The travelling salesman problem with pick-up and delivery. *European Journal of Operational Research* **79** 299–310.

Or, I. 1976. Traveling salesman-type combinatorial problems and their relation to the logistics of blood banking, PhD Thesis, Northwestern University, Evanston.

Parragh, S. N., Doerner, K. F., Hartl, R. F. (2008), A survey on pickup and delivery problems. Part I: Transportation between customers and depot. *Journal für Betriebswirtschaft* **58** 21–51.

Parragh, S. N., Doerner, K. F., Hartl, R. F. (2008), A survey on pickup and delivery problems. Part II: Transportation between pickup and delivery locations. *Journal für Betriebswirtschaft*, **58** 81–117.

Rego, C., D. Gamboa, F. Glover, C. Osterman. 2010. Traveling Salesman Problem Heuristics: Leading Methods, Implementations and Latest Advances. *European Journal of Operational Research*, In Press, DOI: 10.1016/j.ejor.2010.09.010.

Savelsbergh, M. W. P., M. Sol. 1995. The general pickup and delivery problem. *Transportation Science* **29** 17–29.

Wang, F., A. Lim, Z. Xu. 2006. The one-commodity pickup and delivery travelling salesman problem on a path or a tree. *Networks* **48** 24–35.

Zhao, F., S. Li, J. Sun, D. Mei. 2009. Genetic algorithm for the one-commodity pickup-and-delivery traveling salesman problem. *Computers & Industrial Engineering* **56** 1642–1648.

Zhao, F., J. Sun, S. Li, W. Liu. 2009. A hybrid genetic algorithm for the traveling salesman problem with pickup and delivery. *International Journal of Automation and Computing* **6** 97–102.