Available online at www.sciencedirect.com

ScienceDirect

Computers & Operations Research 34 (2007) 2743–2757

computers & operations research

www.elsevier.com/locate/cor

An efficient variable neighborhood search heuristic for very large scale vehicle routing problems

Jari Kytöjoki[a], Teemu Nuortio[b], Olli Bräysy[a], Michel Gendreau[c,*]

[a]Agora Innoroad Laboratory, Agora Center, University of Jyväskylä, P.O. Box 35, FI-40014, Finland
[b]Department of Environmental Sciences, University of Kuopio, P.O. Box 1627, FI-70211 Kuopio, Finland
[c]Center for Research on Transportation, University of Montreal, P.O.Box 6128, Succursale Centre-ville, Montreal, H3C 3J7, Canada

Available online 13 December 2005

Abstract

In this paper, we present an efficient variable neighborhood search heuristic for the capacitated vehicle routing problem. The objective is to design least cost routes for a fleet of identically capacitated vehicles to service geographically scattered customers with known demands. The variable neighborhood search procedure is used to guide a set of standard improvement heuristics. In addition, a strategy reminiscent of the guided local search metaheuristic is used to help escape local minima. The developed solution method is specifically aimed at solving very large scale real-life vehicle routing problems. To speed up the method and cut down memory usage, new implementation concepts are used. Computational experiments on 32 existing large scale benchmarks, as well as on 20 new very large scale problem instances, demonstrate that the proposed method is fast, competitive and able to find high-quality solutions for problem instances with up to 20,000 customers within reasonable CPU times.
© 2005 Elsevier Ltd. All rights reserved.

Keywords: Vehicle routing; Heuristics; Variable neighborhood search; Guided local search; Large scale problems

1. Introduction

Efficient distribution of goods is of paramount importance, not only for the survival of many logistic service providers, but ultimately for the competitiveness of a region's economy by lowering the cost of goods to consumers. Cost savings can be achieved, in particular, through the use of high quality routes and schedules for the fleet of vehicles that performs distribution tasks.

The vehicle routing problem (VRP), introduced by Dantzig and Ramser [1], holds a central place with regards to the determination of efficient routes in distribution management. Consequently, it has become one of the most widely studied problems in combinatorial optimization. The classical VRP can be formally defined as follows. Let $G=(V, E)$ be a connected digraph where V is a set of $n+1$ nodes and E a set of arcs with non-negative weights and associated travel times. One of the nodes represents a depot where the fleet of vehicles is based. With each node i, apart from the depot, is associated a demand q_i that can be a delivery from or a pickup to the depot. The problem consists in determining a set of routes starting and ending at the depot such that the total distance traveled is minimized, while the cumulative demand of the customers serviced by any route t, $\sum_{i\in t} q_i$, does not exceed the vehicle capacity Q and its

* Corresponding author. Tel.: +1 514 3437435; fax: +1 514 3437121.
E-mail address: michelg@crt.umontreal.ca (M. Gendreau).

duration D_t does not exceed a preset upper limit L. All problem parameters are assumed to be known with certainty. Moreover, each customer must be served by exactly one vehicle, thus prohibiting split service and multiple visits.

The VRP is a NP-hard problem [2]. This property makes it difficult to solve the VRP to optimality. Exact algorithms have been used to solve problems with up to 50 customers in a reasonable computing time. For more information on exact solution approaches for the VRP, we refer the reader to the surveys of Toth and Vigo [3], Naddef and Rinaldi [4], and Bramel and Simchi-Levi [5].

For larger problems, heuristics are more appropriate. Basically, VRP heuristics can be divided into three categories: construction heuristics, improvement heuristics and metaheuristics. Construction heuristics generate feasible solutions by performing sequences of simple operations that minimize or maximize a given criterion. This class of methods includes the well-known savings heuristic of Clarke and Wright [6] that proceeds by merging two routes into a single route according to the savings obtained by this merger. Other well-known examples of construction heuristics are the sweep algorithm of Gillett and Miller [7], the insertion heuristic of Christofides et al. [8], the improved savings heuristic of Paessens [9] and the petal algorithm of Renaud et al. [10]. More details can be found in the survey papers of Laporte et al. [11], Laporte and Semet [12], and Van Breedam [13], which presents a parametric analysis of a number of construction heuristics.

Improvement heuristics seek to iteratively enhance a feasible solution (usually generated by a construction heuristic) through locally replacing a set of arcs or customer nodes. These local operations are performed as long as they produce improvements in the value of the objective function. These methods thus typically yield solutions that are local optima, with respect to the set of solution modifications considered. For more information, we refer the reader to Lin [14], Or [15], Thompson and Psaraftis [16], Van Breedam [17], Kindervater and Savelsbergh [18], Ergun et al. [19], and the surveys by Laporte et al. [11] and Laporte and Semet [12].

During the last decade, most of the research efforts aimed at solving the VRP have focused on the development of various metaheuristic algorithms. A metaheuristic can be defined as a top-level general strategy which guides other heuristics to search for good solutions. Most of the VRP metaheuristics are based on some construction and improvement heuristics, i.e., they use the so-called local search principle [20]. Well-known examples of this class of metaheuristics are simulated annealing and tabu search. For efficient implementations of tabu search in the VRP context, see Osman [21], Taillard [22], Gendreau et al. [23], Rochat and Taillard [24], Xu and Kelly [25], Barbarosoglu and Ozgur [26], Cordeau et al. [27], and Toth and Vigo [28]. Tarantilis et al. [29,30] and Li et al. [31] investigate variants of simulated annealing. The second main metaheuristic principle is population search, which consists of maintaining a pool of solutions, and efficiently combining and replacing the solutions in the pool. A classical example of population search is the genetic algorithm. For recent successful applications to the VRP, see Berger and Barkaoui [32], Prins [33], and Baker and Ayechew [34]. Mester and Bräysy [35] adapt a variant of genetic algorithm, evolution strategies, for the VRP and combine it with guided local search. An extension of the genetic algorithm, the adaptive memory concept, is applied in Rochat and Taillard [24], Tarantilis and Kiranoudis [36] and Tarantilis [37]. The ant algorithms of Bullnheimer et al. [38,39], and Reimann et al. [40] can also be viewed as a population search metaheuristics. Baker and Carreto [41] study a greedy randomized adaptive search procedure (GRASP) and Van Breedam [42] presents an in-depth comparison among a set of improvement heuristics and metaheuristics. For more details on the metaheuristics for the VRP, we refer to extensive surveys of Laporte et al. [11], Gendreau et al. [43], Cordeau et al. [20,44], and Golden et al. [45].

During the last 10 years, a lot of research attention has been devoted in solving VRPs with up to nearly 500 customers with various heuristics. Recently, Li et al. [31] have proposed a set of 12 new benchmarks consisting of problems with 560–1200 customers. The authors use the term "very large scale" in referring to these instances. In practice there are, however, vehicle routing problems of a much larger size. For example, in the context of waste collection and mail and newspaper delivery, some problems can involve thousands of customer points, and sometimes even several tens of thousands customers. In the past, these problems have traditionally been modeled as arc routing problems [46]. However, nowadays real-time information is available, e.g., on the amount of waste in each separate container and on newspaper subscription status. Due to recycling, different types of waste must be treated separately and collected by particular collection trucks on specific days. The need to provide high-quality service level also in sparsely populated areas is continuously increasing. Thus, now there is a need to model the above described problems using a more detailed node routing approach instead of an arc routing one. In addition, because of the continuously increasing collaboration among transportation companies and other logistics partners, both at the domestic and the international level, VRP instances are becoming all the time larger and more complex. There is a clear need to develop solution procedures that are able to solve these very large scale problems. To the best of our knowledge, the largest VRPs for which feasible solutions have been reported involve only 1200 customers. The main contribution of this paper is to develop a set of 20 problems with 2400–20,000 customers and new efficient metaheuristic solution procedure for their solution. The developed metaheuristic uses the variable neighborhood search (VNS) strategy [47] to guide a set of seven standard improvement heuristics; this strategy is combined with the well-known guided local search (GLS) [48,49] metaheuristic. Because of the large size of the considered problems, definite efforts have been in the implementation details to keep the computational workload and the memory requirements within reasonable limits. Computational experiments on the large scale benchmark problems of Golden et al. [45] and Li et al. [31], as well as on the 20 new very large scale instances, demonstrate that the proposed method is fast and competitive, and able to find high-quality feasible solutions for problems up to 20,000 customers on a standard PC in a reasonable amount of time.

The remainder of this paper is organized as follows. The proposed metaheuristic is described in Section 2, along with some details on the detailed techniques used for its implementation. Computational results are reported in Section 3, which includes a description of the 20 new very large scale problems. Section 4 concludes the paper.

2. The problem solving methodology

The proposed VNS metaheuristic consists of two phases. In the first phase, an initial solution is created by a cheapest insertion heuristic that is hybridized with a set of seven improvement heuristics. These improvement heuristics are applied at various stages of this construction phase. In the second phase, an attempt is made to improve the initial solution by applying the same seven improvement heuristics according to the variable neighborhood descent (VND) strategy of Mladenovic and Hansen [47] until no more improvements can be found. To facilitate escaping local minima, moves that worsen the solution value are also accepted by penalizing certain solution features as in guided local search. Therefore, the proposed new method is called guided variable neighborhood search (GVNS).

2.1. The improvement heuristics

Before proceeding to a detailed description of the global heuristic, we first recall the main features of the improvement heuristics that it will control and guide. As we already mentioned, seven improvement heuristics are used; three of these are “intra-route” operators that work on a single route at the time (2-opt [50], Or-opt [15] and 3-opt [14]), while the four others are “inter-route” operators that modify several routes simultaneously (exchange, relocate [51], 2-opt* [52] and CROSS-exchange [53]).

The 2-opt heuristic tries to improve a single route by replacing two of its arcs by two other arcs and iterates until no further improvement is possible. Correspondingly, in 3-opt three arcs are replaced simultaneously in order to improve the solution. The Or-opt is closely related. The basic idea is to relocate a chain of l consecutive customers. This is achieved by replacing three arcs in the original tour by three new ones without modifying the orientation of the route.

The exchange heuristic swaps simultaneously the position of two customers in two different routes, while the relocate operator simply reinserts a single customer at a time in an alternate route. 2-opt* combines two routes so that the last customers of a given route are introduced after the first customers of another route, thus preserving the orientation of the routes. In other words, the end portions of two routes are swapped. In CROSS-exchanges, two arcs, respectively $(i-1, i)$ and $(k, k+1)$, and $(j-1, j)$ and $(l, l+1)$, are first removed from each of the two routes considered; then the segments i–k and j–l, which may contain an arbitrary number of customers, are swapped by introducing new arcs $(i-1, j)$, $(l, k+1)$, $(j-1, i)$ and $(k, l+1)$.

2.2. The construction phase

The algorithm starts with a step in which the geographical structure of the problem instance is determined. This step is required, because the algorithm uses different parameter settings for instances that display a different structure. The objective here is to check whether the depot is located close to the center of gravity of the demand points and if these are positioned regularly around the depot. If this is the case, the instance is deemed to be “symmetric”, otherwise, it is considered to be “asymmetric”. More precisely, the algorithm checks if the depot coordinates correspond to the average of the coordinate values of the demand points, if the customers are located on concentric rings around the depot, and if the number of the customers on each ring and their total demand are equal. An instance is considered to be "symmetric" only if all these conditions hold.

The initial solution is then created by constructing routes sequentially one at a time until all customers have been routed. A route is first initialized with a "seed" customer and unrouted customers are then added one by one to this route. In contrast to previous research, we terminate the construction of a route before the route is 100% full with respect to the scheduling horizon and/or capacity constraint, even though more customers could be inserted in it. The rationale is that the improvement heuristics work a lot better if there is some free space in the routes. If unrouted customers remain after constructing a route, the initialization and insertion procedures are repeated until all customers are serviced. In the case of asymmetric problems, the seed customers are selected by finding the unrouted customer that is geographically closest to the depot. For symmetric instances, the traditional strategy of selecting the farthest customer from the depot as the next seed is used. When two or more customers are equally close to the depot, the customer with the largest demand is selected.

The insertions are based on the distance criterion only, i.e., the heuristic finds an unrouted customer u that least increases the distance of the incumbent route and inserts that customer between two adjacent customers i and j on the route. After the insertion of each group of k customers in the route currently under construction, an attempt is made to reorder the route with the intra-route improvement heuristics. Two different values of k were determined experimentally, depending on the geographical structure of the problem.

These procedures are applied with the first-accept strategy and according to the VND scheme: 2-opt is first applied until no more improvements can be found; the algorithm then proceeds with Or-opt until no more improvement can be found; finally, 3-opt is applied in the same fashion. If any of the three improvement heuristics was able to improve the route, the loop is restarted again with 2-opt. The procedure is repeated until no more improvements can be found with any of the three heuristics.

Once the incumbent route is 100-Δ% full with respect to the scheduling horizon or the capacity constraint, an attempt is made to improve the current partial solution by applying a VND procedure with the four inter-route improvement heuristics described to all two-route combinations involving the route just created. These heuristics are applied with the first-accept strategy in a way similar to the one used for the intra-route operators: exchange is first applied until no more improvements can be found; the algorithm then switches to the relocate, 2-opt* and CROSS-exchange heuristics, in this order. The reason for performing exchange before relocate is to make more free space for both routes thus yielding a somewhat more balanced treatment of them in the following. If any improvement is found by this inter-route VND procedure, at its end, an attempt is made to reorder both associated routes separately with the above intra-route VND. When all customers have been routed and no more improvements can be found, the second phase is launched.

2.3. The improvement phase

In the second phase, an attempt is made to further improve the feasible solution obtained by guiding the seven improvement heuristics with the proposed GVNS metaheuristic. The main difference between the VNS applied in the first phase and the VNS applied in the second phase is that in the first phase the focus is on the route currently under construction. In particular, inter-route exchanges are attempted only between this route and previously constructed routes, but not between pairs of previously constructed routes. In the second phase, inter-route exchanges are tested for all two-route combinations. For each route pair, we first apply the inter-route VND step as in the first phase and if an improvement is found in this step, the intra-route procedures are applied to the two routes being considered. This is followed by a step in which a GLS-type strategy is applied to this pair of routes. This step involves repeating the inter-route VND procedure with a modified objective function that penalizes long arcs included in the current pair of routes with the goal of forcing them out of the routes.

More precisely, the first arc to be penalized is the longest one. Its length is penalized by a adding to it a user-defined value λ, which depends on the geographical structure of the instance. Both the inter- and the intra-route VND procedures are then re-applied until they eventually end up in a (possibly new) local minimum with respect to the heuristics. A new arc is then selected for penalization. The arc chosen for penalization is the one for which the highest value of the following "utility" function is achieved:

$$U = \frac{\lambda \times p_{ij} + c_{ij}}{p_{ij} + 1}, \tag{1}$$

where c_{ij} is the length of arc (i, j) and p_{ij} is the number of times arc (i, j) has been penalized during the consideration of the current pair of routes. Again, its cost is increased by λ. At this point, a check is made to determine if the step should be terminated by comparing the current total of the penalty factors applied for the step, i.e., $\sum p_{ij} \times \lambda$, to a threshold τ which is equal to λ times the length of the first arc penalized in this step. If $\sum p_{ij} \times \lambda > \tau$, the step is terminated, otherwise both the inter- and the intra-route VND procedures are executed again with the modified objective function

$$h(s) = g(s) + \lambda \sum p_{ij}, \tag{2}$$

where s denotes a candidate solution and $g(s)$ is the original objective function. This procedure is applied repeatedly until the termination criterion is met. Note that during this GVNS step, the best feasible solution is actually modified only if an improvement is found with respect to the original objective function.

This strategy is reminiscent of the threshold accepting (TA) metaheuristic introduced by Dueck and Scheurer [54]. In contrast to TA, the threshold value is used to limit the total allowed worsening for the current pair of routes, and the threshold is computed by the algorithm instead of being a user-defined parameter that is varied according to a specific cooling scheme. It should also be stressed that the threshold is recalculated every time a new pair of routes is selected. During computational experiments, it was observed that, in most cases, only a few arcs were penalized for each pair of routes.

2.4. Pseudo-code of the proposed algorithm

The following pseudo-code summarizes the different steps of the proposed algorithm:

```
Phase 1:
Step 1.1: Determine the geographical structure of the problem, $S$.
Step 1.2: Set $r = 0$, $\Delta = 2$; set $k$ and $\lambda$ according to the value of $S$.
Step 1.3: Until all customers have been routed
Step 1.3.1: Determine a seed customer $s$ by finding the unrouted customer farthest from the depot if $S$=symmetric, and the unrouted customer closest to the depot otherwise. Insert this seed customer in route $r$. For each unrouted customer $i$, compute the cost of inserting it in position depot-$s$, $C'_i = c_{0i} + c_{is} - c_{0s}$ or in position $s$-depot, $C''_i = c_{si} + c_{i0} - c_{s0}$. For each of the two insertion positions, sort the insertion costs in ascending order, and store in a priority queue associated with it the 1000 smallest insertion costs and the corresponding customers.
Step 1.3.2: Set $j = 0$.
Step 1.3.3: While
    $\sum q_i < \frac{100 - \Delta}{100} \times Q \quad \text{and} \quad D_j < \frac{100 - \Delta}{100} \times L.$
Step 1.3.4: Find in memory (from the priority queues) the unrouted customer $i_{\min}$ with the smallest insertion cost and such that after its insertion in route $r$ the condition
    $\sum q_i < \frac{100 - \Delta}{100} \times Q \quad \text{and} \quad D_j < \frac{100 - \Delta}{100} \times L$
will still be verified. If such a customer cannot be found, goto Step 1.3.8.
Step 1.3.5: Insert $i_{\min}$ into its best position between customers $l$ and $m$ in $r$. Set $j = j + 1$. Compute the insertion costs for all unrouted customers $i$ in position $l - i_{\min}$ of route $r$, $C'_i = c_{li} + c_{i i_{\min}} - c_{l i_{\min}}$. Sort these insertion costs in ascending order, and store the 1000 smallest in the existing priority queue of $l$-$m$, which now corresponds to insertion position $l - i_{\min}$. Similarly, compute the insertion costs for all unrouted customers $i$ in position $i_{\min} - m$, $C''_i = c_{i_{\min} i} + c_{im} - c_{i_{\min} m}$, sort them and store the 1000 smallest in a new priority queue for position $i_{\min} - m$.
Step 1.3.6: If $j \bmod k = 0$, apply the 2-opt, Or-opt and 3-opt heuristics to $r$ until no more improvements can be found.
Step 1.3.7: Goto Step 1.3.4.
Step 1.3.8: Apply the exchange, relocate, 2-opt* and CROSS-exchange heuristics to all route pairs $\{r, t\}$, where $t < r$ until no more improvements can be found. In case an improvement is found, apply 2-opt, Or-opt and 3-opt heuristics to $r$ and $t$ until no more improvements can be found. Set $r = r + 1$ and goto Step 1.3.
```

Phase 2:

Step 2.1: Until no more improvements can be found for any pair of routes.

Step 2.1.1: Until all route pairs have been considered, select the next route pair $\{t_1, t_2\}$, where $0 \leqslant t_1 \leqslant r$ and $0 \leqslant t_2 \leqslant r$ and $t_1 \neq t_2$.

Step 2.1.2: Set the threshold $\tau = \lambda \times \max_{(i,j)\in(t_1,t_2)}\{c_{ij}\}$ and set $P_{ij} = 0$ for all arcs in $\{t_1, t_2\}$.

Step 2.1.3: Apply exchange, relocate, 2-opt* and CROSS-exchange to route pair $\{t_1, t_2\}$ until no more improvements can be found. If an improvement has been found, apply the 2-opt, Or-opt and 3-opt heuristics to both t_1 and t_2 separately until no more improvements can be found.

Step 2.1.4: Select an arc to be penalized according to equation (1) and update. If $\sum p_{ij} \times \lambda > \tau$, goto Step 2.1.1. Repeat Step 2.1.3 evaluating moves according to (2). Repeat Step 2.1.4.

2.5. *Implementation techniques*

In this section, we shortly describe the implementation techniques that we used to speed up the algorithm and reduce memory usage.

Our algorithm considers both the distance and the travel time for each (directed) pair of nodes. A straightforward approach to this issue would be to maintain distinct distance and travel time matrices, but this would cause problems when extremely large problems, e.g., 30,000-customer problems, are considered. Indeed, in such a case, the amount of memory required to store both matrices (7.2 billions bytes) would exceed the maximum amount of memory addressable using a standard 32-bit architecture, i.e., 4 GB. A first step towards reducing memory usage consists in encoding both distance and travel time for each pair of nodes in a single word, thus cutting down memory requirements by half. In fact, to maintain sufficient precision in the values, it is more effective to maintain a somewhat different representation that replaces the travel time between two nodes by the average speed between them, since it is possible to encode speed with sufficient precision using just a few bits, thus leaving more bits for encoding the distance. In a standard 32-bit *float* type representation of floating point numbers with 23 bits of mantissa [55], the 6 or 7 least significant bits of the mantissa can be used to encode the speed while 16 or 17 for the distance, which is sufficiently precise. In our implementation, we went one step further and decided to not record distances directly, but rather using "twisty-road-curve factors" [59], based on a histogram analysis and re-weighting of the input data, that give the ratio between actual distances and Euclidean distances. Thus, for each pair of nodes, we record the associated twisty-road-curve factor and the speed, which can be represented together in a compact way using only 8 or 16 bits (the actual choice of representation depends on how close the real distance values are to the Euclidean values). Using this representation requires additional computations, because distances and travel times must be recomputed whenever they are needed, but since modern processors execute calculations much faster than they perform most memory operations [56–58], this is not a big drawback. As a result of these implementation decisions, much larger real-life problems can be solved with a very reasonable amount of memory (in the case of theoretical benchmark problems with Euclidean distances and uniform speeds, the proposed encoding mechanism is not relevant). For example, in the case of 20,000 customer problems, one would normally need $20{,}000 \times 20{,}000 \times (\text{distance} + \text{time}) \times 4\,\text{bytes} = 2.98$ GB memory. Using the proposed memory reduction techniques, only 0.186–0.3725 GB memory is needed, depending on the case. More details on the implementation of the algorithm can be found in Kytöjoki [59].

To speed up the computations of seed customers, all customers are ordered at the beginning according to their distance from the depot. The next seed is then determined by finding the first unrouted customer in this ordered list.

To limit the computational effort in the initial construction heuristic, we only consider for insertion into the current route the unrouted customers with the smallest insertion costs. More precisely, for each potential insertion position between two adjacent customers of the route, we record in a priority queue the 1000 smallest insertion costs and the corresponding customer; other customers are ignored (obviously, if less than 1000 unrouted customers remain, we only record insertion costs for these remaining customers). Two such queues are created after the selection of the seed customer of a route. Then, every time a customer is inserted in the route, two priority queues of 1000 customers with their associated insertion cost are set up for the two new potential insertion positions (in fact, one of these corresponds to the queue previously used for the position in which the insertion was just performed). Note that the other priority queues are not updated at that time. It is therefore possible, when constructing very long routes (with more than 1000 customers), that one of the priority queues might become empty when customers already routed on the current route

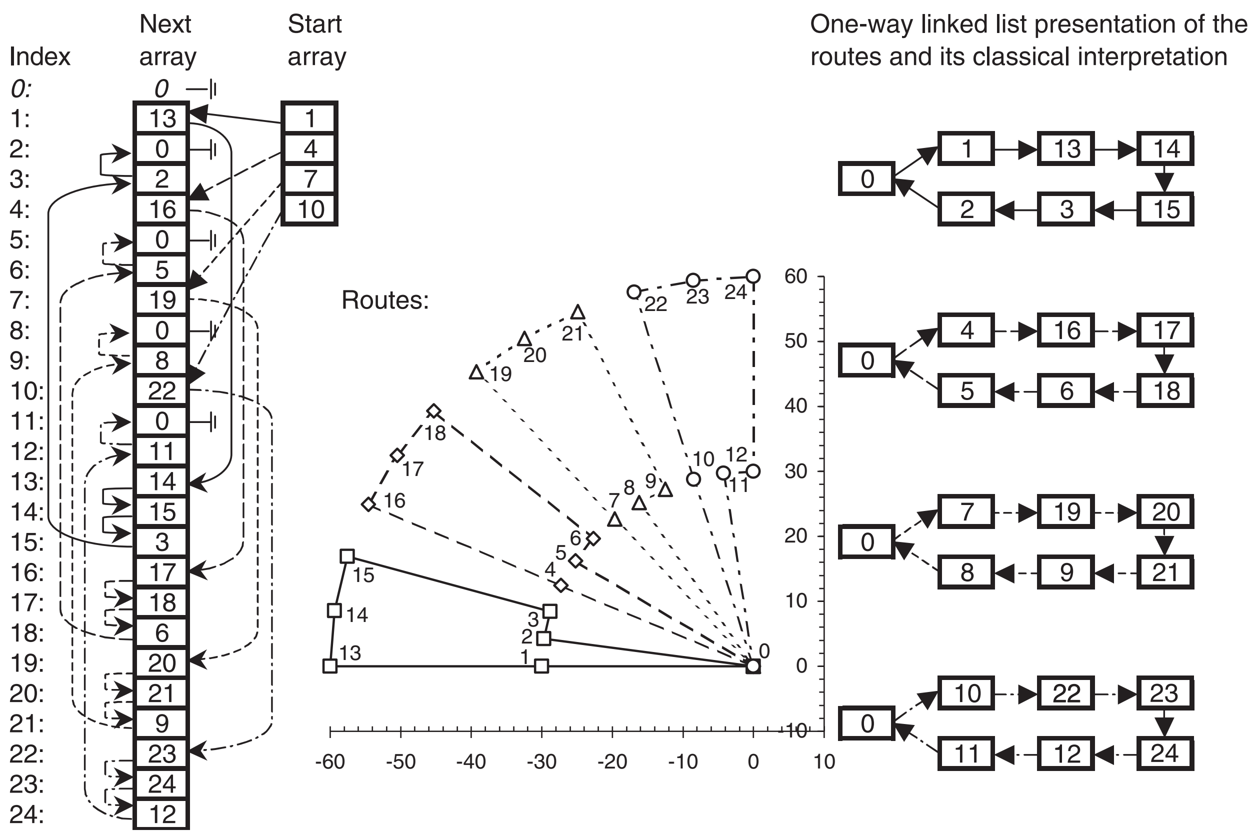
Index
Next array
Start array
One-way linked list presentation of the routes and its classical interpretation
Routes:

Fig. 1. Example of the one-way linked list data structure in an array (depot 0 at (0, 0)).

Index:	0:	1:	2:	3:	4:	5:	6:	7:	8:	9:	10:	11:	12:	13:	14:	15:	16:	17:	18:	19:	20:	21:	22:	23:	24:
Next:	-1	13	-4	2	16	-7	5	19	-10	8	22	0	11	14	15	3	17	18	6	20	21	9	23	24	12

Fig. 2. An alternative way to store the solution information.

are popped out. Should this occur, this queue is reinitialized with the next 1000 smallest insertion costs for yet unrouted customers. With this strategy and the priority queues, it is possible to quickly determine the cheapest insertion and its cost.

To further speed up the computations, we use a one-way linked list data structure in an array (called "next array") to represent solutions (see Fig. 1 for an example with four short routes). This structure combines the advantages of the well-known linked list and array data structures. As in one-way linked lists, information on the following customer for each customer is preserved. However, here the information is stored into an array whose size is equal to the number of the customers. The first customer of each route is recorded in a separate auxiliary array ("start array"). These two arrays are enough to completely represent solutions. Using this structure, changes to the solution can be performed very quickly and in a constant time, without having to loop as in standard linked list implementations. For example, the addition of a new customer *u* between two adjacent customers *i* and *j* is done simply by changing the "next-values" of *u* to *j* and *i* to *u*. Similarly, one can delete a customer or reverse part of a route very quickly.

It is possible to omit the start array and do everything with a single array, but this would cause additional complexity in the code implementation. A simple example showing how to store all the route information in the case of Fig. 1 is presented in Fig. 2, where non-positive values (starting from index 0) give the beginning of the next route index (as a negative index value) and 0 refers to the end of the routes. For a more detailed description, we refer the reader to Kytöjoki [59].

Compared to the one-dimensional tabular representation of solutions typically used, our approach is much quicker. For example, in order to add a new customer in a tabular representation, one must change the position index of all customers following the current insertion position, resulting in computations of $O(n)$ complexity. We are not aware of any previous paper reporting the use of this or a similar data structure in the VRP context.

3. Experimental results

Computational experiments were performed to assess the performance of the proposed algorithm. We now describe the test data sets, as well as the parameter values used, before presenting the results obtained with our metaheuristic and analyzing them.

3.1. Problem data and experimental setting

The proposed heuristic algorithm has been implemented in Visual C++ 6.0 and tested on an AMD Athlon64 3000+ (1 GB RAM) computer. The computational experiments were performed on three separate data sets: the 20 large scale benchmark problems of Golden et al. [45], 12 very large benchmark problems of Li et al. [31], and 20 new very large scale benchmark problems introduced in this paper. Travel times between two customers correspond to their relative Euclidean distance in all problems. As the applied algorithm is deterministic, it was sufficient to perform only a single run of the algorithm for each problem.

The 20 large scale benchmarks of Golden et al. [45] consist of 200–483 customers. The first eight instances have route-length restrictions. Each problem has a geometric structure: eight problems have customers located in concentric circles around the depot, four problems have customers located in concentric squares with the depot located in one corner, four problems have customers located in concentric squares around the depot, and four problems have customers located in a six-pointed star around the depot.

The problems of Li et al. [31] have 560–1200 customers and route length restrictions, and they exhibit geometric structure (concentric circles around the depot) that allows a user to visually estimate a high-quality solution.

The 20 new very large scale problems can be divided into three groups. The first group of 12 problems is similar to Li et al. [31] instances, and it was generated with the problem generator of Li et al. [31] by setting the parameter values such that larger problems consisting of 2400–20,000 customers were obtained. The problem parameters are given in Table 1.

The second group of four problems was extracted from a real-life waste collection application in Eastern Finland. There are, respectively 7798, 9516, 8454 and 10,217 customers in the problems noted as W(est), E(ast), S(outh) and M(iddle). As the problem names imply, the problems were generated by extracting four geographical regions from the whole collection area according to the x and y coordinates of the waste bins. The division was done in such a way that the areas have approximately an equal number of customers. The demand for each customer was determined according to the actual average waste weight values, assuming that only the two most commonly used container types are in

Table 1
The parameters used for the problem generator of Li et al. [31]

Problem	A	B	n	Capacity	Max route length
33	40	60	2400	4900	7800
34	40	90	3600	7300	11,700
35	40	150	6000	12,100	19,500
36	40	180	7200	14,500	23,400
37	40	210	8400	16,900	27,300
38	40	240	9600	19,300	31,100
39	40	270	10,800	21,700	35,100
40	40	300	12,000	24,100	39,000
41	40	330	13,200	26,500	42,900
42	40	360	14,400	28,900	46,800
43	40	420	16,800	33,700	54,500
44	40	500	20,000	40,100	64,800

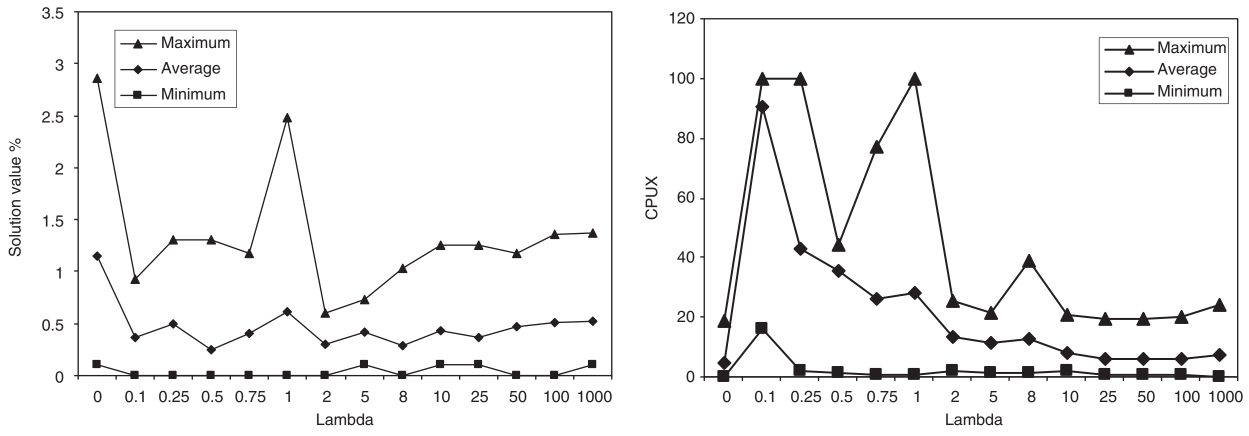

Fig. 3. The sensitivity of the total distance and CPU time with respect to parameter λ.

use. The vehicle capacity (26 metric tons) is set equal to the capacity of the typical waste collection truck in Finland. There are both clustered and randomly distributed waste containers in all instances, and there is only one depot for all problems, located in the periphery of the western area.

The third group consists of four randomly created problems with a centrally located depot. There are, respectively 3000, 6000, 9000 and 12,000 customers in these instances noted R3, R6, R9 and R12. The problems were created by determining the x and y coordinates of the customers randomly in the range 0–10,000. The demands were determined randomly to allow each vehicle (with capacity of 5000 units) to serve between 10 and 30 customers, and there are on the average 15 customers on each route, which is typical for many real-life delivery applications. In contrast with the second (waste collection) problem group where several hundreds of containers can be collected by a vehicle on a single route, these random problems are more representative of a "short-haul" delivery environment with shorter scheduling horizon and more (several hundreds) vehicles. For problem R12, it turns out that as many as 812 vehicles are required to service all customers according to the solution obtained. These problems are available at www.top.sintef.no.

The sensitivity of the results to the value of parameter λ is illustrated in Fig. 3 using a randomly selected set of test problems. More precisely, we selected randomly 30% of the problems separately from each of the five test sets described above. We experimented with values of λ in the range 0–1000. Here one must note that when $\lambda = 0$, the GLS is not used at all, and the results correspond to the VNS only. In Fig. 3, the values of λ are listed along the x-axis. In the left part of the figure, the three curves illustrate the difference in percents between the best solution found and the worst, average and the best solution quality obtained with the corresponding λ, whereas in the right part of the figure, the curves illustrate the CPU time needed as a percentage of the longest CPU time observed.

As can be seen from the figure, differences in terms of solution quality are small and several λ-values provide the best solution quality. In general, values 0.5 and 8 seem to work best. A more detailed analysis indicated that $\lambda = 0.5$ worked best for symmetrical problems and $\lambda = 8$ for other problems. Thus, the remaining computational experiments were conducted using these values. It is interesting to note that CPU times are much more sensitive to the value of λ than the solution values. The reason for the variance of the CPU times lies in the threshold limit used for the worsening of the solution values. The relation between the calculated threshold and the value of λ determines how often arcs are penalized during the search.

The values for parameter k that controls the frequency of improvements within the initial solution construction were also determined experimentally using the same randomly selected problem set as above. The results are illustrated in Fig. 4 that has the same format as Fig. 3. When $k = 0$, the intra-route optimization is not applied. According to the figure, the differences appear small in terms of both solution quality and computing time. In general and for symmetric problem instances, $k = 4$ appeared best and for other problems we used the value $k = 64$. The rationale for more frequent improvements in the case of symmetric problems is that for symmetric problems large changes to solution structure appear harder, which emphasizes the significance of the initial solution.

For the free space Δ left in the routes when constructing the initial solution, we experimented with values ranging from 0 to 5% using the above random problem set and noticed that in most cases $\Delta = 2\%$ is enough to facilitate the functioning of the improvement heuristics. Therefore, all computational experiments were performed using this value.

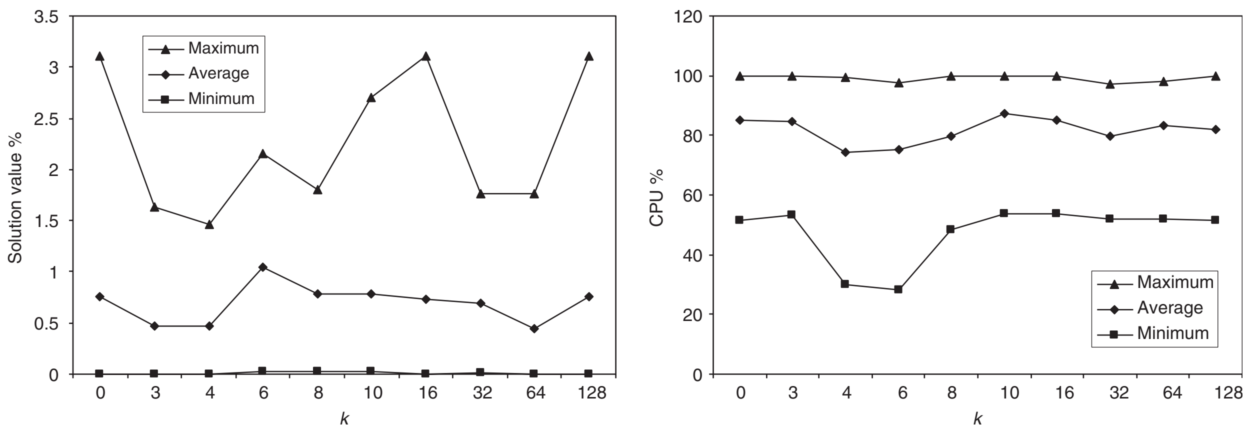

Fig. 4. The sensitivity of the total distance and CPU time with respect to parameter *k*.

3.2. *Results for the Golden et al. [45] benchmarks*

The results for the 20 benchmark problems of Golden et al. [45] are presented in Table 2. We consider in Table 2 only a limited set of results of recent metaheuristics that have displayed the best performance according to the literature.

The first row lists the authors, and the next 20 rows show the total distance values for each author for the 20 problems. The last three rows report the average deviation from the best-known solutions in percentage, the type and speed in MHz of the computers used, and the average CPU time per problem in minutes. The first two columns give the problem number and size and the number of vehicles in our solutions. The best-known solution values are taken from Mester and Bräysy [35] and given in the rightmost column. Our results and computation times in seconds are shown in the

Table 2
Comparison of results for the large scale benchmarks of Golden et al. [45]

Problem	Vehicles	Toth and Vigo [28]	Reimann et al. [40]	Li et al. [31]	Mester and Bräysy [35]	VNS	CPU (s)	GVNS	CPU (s)	Best known
1 (240)	10	5736.15	5644.02	5666.42	5627.54	5867.84	0.08	5867.84	0.26	5627.54
2 (320)	11	8553.03	8449.12	8469.32	8447.92	8476.26	0.19	8476.26	0.4	8447.92
3 (400)	10	11,402.75	11,036.22	11,145.80	11,036.22	11,043.41	0.29	11,043.41	0.66	11,036.22
4 (480)	10	14,910.62	13,699.11	13,758.08	13,624.52	13,638.37	0.49	13,631.72	1.59	13,624.52
5 (200)	5	6697.53	6460.98	6478.09	6460.98	**6460.98**	0.09	**6460.98**	0.15	6460.98
6 (280)	7	8963.32	8412.90	8539.61	8412.88	8415.67	0.15	8415.67	0.29	8412.80
7 (360)	9	10,547.44	10,195.59	10,289.72	10,195.56	10,297.66	0.31	10,297.66	0.89	10,195.56
8 (440)	11	12,036.24	11,828.78	11,920.52	11,663.55	11,876.10	0.40	11,872.64	1.38	11,663.55
9 (255)	14	593.35	586.87	588.25	583.39	620.67	0.08	620.67	0.77	583.39
10 (323)	16	751.66	750.77	749.49	742.03	800.93	0.12	784.77	1.68	742.03
11 (399)	18	936.04	927.27	925.91	918.45	1001.09	0.16	986.80	1.78	918.45
12 (483)	19	1147.14	1140.87	1128.03	1107.19	1220.69	0.26	1209.02	2.77	1107.19
13 (252)	26	868.80	865.07	865.20	859.11	945.28	0.07	925.81	0.45	859.11
14 (320)	30	1096.18	1093.77	1097.78	1081.31	1183.40	0.12	1155.19	0.67	1081.31
15 (396)	33	1369.44	1358.21	1361.41	1345.23	1476.56	0.16	1461.49	1.08	1345.23
16 (480)	37	1652.32	1635.16	1635.58	1622.69	1755.63	0.24	1742.86	1.66	1622.69
17 (240)	22	711.07	708.76	711.74	707.79	743.81	0.08	726.01	0.53	707.79
18 (300)	27	1016.83	998.83	1010.32	998.73	1105.43	0.09	1077.53	0.76	998.73
19 (360)	33	1400.96	1367.20	1382.59	1366.86	1455.40	0.13	1444.51	1.04	1366.86
20 (420)	38	1915.83	1822.94	1850.92	1821.15	1989.64	0.18	1938.12	1.38	1821.15
Dev. %		2.87	0.59	1.05	0.00	1.98		1.71		0.00
Computer		P 200	P 900	AMD1G	P 2000	AMD3G		AMD3G		N/A
CPU min.		17.5	49.3	1.1	72.9	0.003		0.017		N/A

Table 3
Computational results for the 12 benchmark problems of Li et al. [31]

Problem	Vehicles	Estimate	VRTR	VRTR best	VNS	CPU (s)	GVNS	CPU (s)	Best known
21 (560)	10	16,212.83	16,602.99	16,602.99	**16,243.93**	0.60	**16,221.22**	4.35	16,212.83
22 (600)	15	14,652.28	14,651.27	14,651.27	14,654.87	0.79	14,654.87	3.46	14,641.64
23 (640)	10	18,801.13	19,005.37	18,838.62	**18,821.47**	0.95	**18,810.72**	6.28	18,801.13
24 (720)	10	21,389.43	21,784.43	21,616.25	**21,401.41**	1.21	**21,401.41**	5.54	21,389.43
25 (760)	21	17,053.26	17,151.43	17,146.41	17,530.89	1.71	17,358.18	13.70	17,053.26
26 (800)	11	23,977.74	24,189.66	24,009.74	24,056.86	1.95	**23,996.86**	9.20	23,977.74
27 (840)	21	18,253.55	17,823.40	17,823.40	18,248.53	1.65	18,233.93	13.40	17,651.60
28 (880)	10	26,566.04	26,606.11	26,606.11	**26,592.05**	2.42	**26,592.05**	7.26	26,566.04
29 (960)	10	29,154.34	29,181.21	29,181.21	**29,166.32**	2.82	**29,166.32**	5.91	29,154.34
30 (1040)	10	31,742.64	31,976.73	31,961.58	**31,805.28**	3.38	**31,805.28**	6.60	31,742.64
31 (1120)	10	34,330.94	35,369.17	35,355.50	**34,352.48**	5.17	**34,352.48**	7.72	34,330.94
32 (1200)	10	36,919.24	37,421.44	37,410.84	**37,025.37**	4.90	**37,025.37**	8.97	36,919.24
Average % above best-known			1.10	0.87	0.51		0.41		0.00
Average computing times (min)			3.16	N/A	0.04		0.13		N/A

The computations of Li et al. were performed with AMD Athlon 1 GHz computer.

VNS, GVNS and CPU/s columns. The results in the VNS column are obtained by setting $\lambda = 0$, i.e., not allowing any worsening of the solutions in the second phase of the algorithm.

According to Table 2, the proposed VNS heuristics are much faster than any previous method. However, the solution quality is not as good as for the best competing methods. The average deviation from the best-known solutions is 1.98% and 1.71% for the VNS and GVNS, respectively.

3.3. Results for the Li et al. [31] benchmarks

The results for the 12 very large scale instances of Li et al. [31] are presented and compared against the results reported in Li et al. [31] in Table 3. The third column presents the calculated estimates for the optimal solution. Otherwise, the format of the table is the same as in Table 2 described above. The estimated optimal solution values and the best-known solution values are taken from Li et al. [31]. The VRTR column reports the best results obtained by Li et al. [31] with their record-to-record travel algorithm using a variable-length neighbor list and a single parameter setting. The VRTR best column describes the best solutions found by Li et al. with different parameter settings. Our solutions are marked in bold in cases where we obtained a better solution than reported in Li et al. [31].

As can be seen from Table 3, both of the proposed two VNS variants are faster than the method of Li et al., even if one takes into account the differences between the computers that were used. The solution values of VNS and GVNS are also better on the average than the solution values reported by Li et al. The GVNS found 0.1% better solutions in terms of total distance on the average compared to the VNS, although at the cost of being 3–4 times slower. Here one must note that GLS does not suit well for circular problems created with the problem generator of Li et al. [31] because very often the longest arcs are in the outermost circles and are part of the optimal solution. The penalization of "wrong" arcs degrades the performance of the GVNS.

3.4. Results for the new very large scale problems

This section describes the results for the new 20 very large scale vehicle routing problems. In Table 4, we report the results for the 12 problems created with the problem generator of Li et al. [31]. As in Table 3, the first three columns depict the problem names, problem sizes and estimates for the optimal solution value. As in Li et al. [31], the estimates were calculated by exploiting the geometric structure and circular shape of the problems and by assuming a fixed number of routes. If the total number of customers, the number of routes, the number of customers in each circle and the distance between the circles are known, it is possible to estimate the optimal solution structure and value using isosceles triangles. For exact equations, see Appendix. The next two columns, noted Insertion and CPU/min, refer to the solution values obtained with the initial cheapest insertion heuristic without any intra- or inter-route local optimizations,

Table 4
Computational results for the created 12 new circular benchmark problems

Problem	Vehicles	Estimate	Insertion	CPU min.	VNS	CPU min.	GVNS	CPU min.
33(2400)	10	75,743.77	76,141.86	0.04	75,754.55	0.20	75,754.55	0.72
34 (3600)	10	114,568.30	115,022.85	0.08	114,579.08	0.48	114,579.08	1.69
35 (6000)	10	192,217.35	192,677.48	0.24	192,228.14	1.65	192,228.14	5.68
36 (7200)	10	231,041.88	231,565.60	0.34	231,052.66	2.68	231,052.66	9.00
37 (8400)	10	269,866.41	270,288.87	0.46	269,877.20	4.18	269,877.20	13.59
38 (9600)	10	308,690.94	309,101.71	0.76	308,702.92	6.74	308,702.92	20.04
39 (10800)	10	347,515.46	347,931.80	0.93	347,537.00	8.93	347,537.00	26.80
40 (12000)	10	386,339.99	386,773.73	1.14	386,350.77	11.11	386,350.77	35.29
41 (13200)	10	425,164.52	425,576.13	1.36	425,175.30	15.15	425,175.30	44.04
42 (14400)	10	463,989.05	464,437.72	1.48	463,999.83	19.52	463,999.83	54.10
43 (16800)	10	541,638.10	542,152.54	2.02	541,648.89	30.43	541,648.89	85.78
44 (20000)	10	645,170.17	645,613.05	2.95	645,180.95	50.76	645,180.95	144.09
Average			0.19%	0.96	0.005%	12.30	0.005%	34.95

Table 5
Computational results for the created eight real-life based and random problem instances

Problem	Vehicles	Insertion	CPU (min.)	VNS	CPU (min.)	GVNS	CPU (min.)
W (7798)	16	5,856,573.61	0.24	4,586,000.51	13.34	4,559,986.36	34.53
E (9516)	17	5,301,047.29	0.44	4,775,466.04	16.57	4,757,566.36	83.85
S (8454)	15	4,123,568.39	0.35	3,342,955.41	17.60	3,333,695.83	56.19
M (10217)	17	3,602,698.01	0.48	3,178,052.21	17.60	3,170,932.21	77.59
R3 (3000)	204	197,986.51	0.16	187,411.32	0.21	186,219.59	4.75
R6 (6000)	406	369,215.89	0.44	355,084.60	0.98	352,701.73	24.40
R9 (9000)	609	538,120.25	1.16	520,896.05	2.70	517,443.40	57.65
R12 (12000)	812	699,630.46	2.74	684,646.39	4.21	680,832.79	108.43
Average		0.00%	0.83	−14.78%	9.29	−15.13%	55.80

and CPU time consumed, respectively. The last four columns give the results and CPU times of the proposed two VNS variants. The last row describes the average percentage deviation of the solution values above the estimated optimal solution and the average CPU times in minutes.

According to Table 4, all three methods find good results very close to the estimated optimum. On the other hand, the differences between the three tested approaches are very small, implying that the created problems are easy to solve. It appears that already the initial solutions of the insertion heuristic are very close to the estimated optimum, and very hard to improve with the proposed metaheuristics. Therefore, based on this data set, comparisons between the different approaches are hard to make, except regarding the computation times that appear reasonable also for the largest instances.

In Table 5, we report results for the four problems extracted from a real-life waste collection application and for four randomly created problem instances. As in Table 4, results for the three different heuristics are given. The last row presents the average improvement in percentage with respect to the initial insertion heuristic and the average computing times in minutes.

In contrast to Table 4, clearly better solutions compared to the insertion heuristic are obtained according to Table 5 if VNS is applied, indicating the harder nature of these eight problems. On the average, GVNS appears to provide the best solution quality, although the difference with VNS is small. One of the main reasons for the good performance of VNS is the free space of 2% left in the routes, facilitating reinsertions and exchanges. By removing the free space, the results would be clearly worse compared to GVNS. The downside of GVNS is that it is significantly slower than VNS.

4. Conclusions

In this paper we have developed an efficient two-phase variable neighborhood search heuristic that is specifically aimed at solving realistic very large scale vehicle routing problems. In the first phase, an initial solution is created with a hybrid cheapest insertion heuristic applying seven improvement heuristics according to a variable neighborhood search scheme. In the second phase, an attempt is made to improve the initial solution with the same VNS approach, but using a different strategy. Computational results on 32 existing large scale benchmark problems, as well as 20 new very large scale instances, show that the proposed method is fast, competitive and capable of finding high-quality solutions for problem instances with up to 20,000 customers within reasonable CPU times.

Acknowledgements

This work was partially supported by the Henry Ford Foundation, the COMAS Graduate School at the University of Jyväskylä, and the Canadian Natural Sciences and Engineering Council. This support is gratefully acknowledged. We would like to thank also Dr. Wout Dullaert for his valuable feedback and the iWaste project financiers and partners, especially the Environmental Informatics research unit at the University of Kuopio, National Technology Agency of Finland (Tekes), Jätekukko Ltd and Ecomond Ltd.

Appendix

We explain here how to compute the distance estimates used in Tables 3 and 4 for the symmetrical problems with circular shape generated with the problem generator of Li et al. [31]. The structure of the optimal solution depends on the number of routes, the number of customers on each ring and on the distance between the rings. For instance, when the number of customers on each ring is 40 and there is only one ring, namely the first innermost ring, the estimate is

$$10\left(2\times 30+3\times 2\times 30\times \sin\left(\frac{\pi}{40}\right)\right)=10\times 30\times 2\times\left(1+3\sin\left(\frac{\pi}{40}\right)\right)=600\left(1+3\sin\left(\frac{\pi}{40}\right)\right).$$

The first term is the number of routes. There are two radiuses, both of length of 30. So the total distance traveled along radiuses for all 10 routes is $10\times 2\times 30$. The second term $3\times 2\times 30\times\sin(\pi/40)$ refers to the total distance traveled along the ring. The distance between two consecutive customers on the ring corresponds to length of the basis of an isosceles triangle whose other edges are of length 30 and with an apex angle equal to $2\pi/40$. Such a triangle can be divided into two right-angled triangles whose apex angle is $2\pi/(2\times 40)$, and thus length of its basis is $2\times 30\times\sin(\pi/40)$. Similarly, for problems with 80 customers located on two rings, each route should serve two groups of four customers on each of the rings and the estimate is

$$600\left(1+3\sin\left(\frac{\pi}{40}\right)\right)+600\left(1+\sin\left(\frac{\pi}{40}\right)\times(3\times 2-1\times 1)\right)=600\left(2+8\sin\left(\frac{\pi}{40}\right)\right).$$

The same strategy can be used up to 320 customers. Starting from 320 points, it is more beneficial to travel in the direction of the radiuses because the points on the outermost rings are too far away from each other. For problems with $n=40k+320$, $k=0,1,2,\ldots$, customers where the distance between the successive rings is 30, the estimate is given by the formula

$$30\left(n-120+(n+720)\sin\left(\frac{\pi}{40}\right)\right).$$

References

[1] Dantzig GB, Ramser JH. The truck dispatching problem. Management Science 1959;6:80–91.
[2] Lenstra J, Rinnooy Kan A. Complexity of vehicle routing and scheduling problems. Networks 1981;11:221–7.
[3] Toth P, Vigo D. Branch-and-bound algorithms for the capacitated VRP. In: Toth P, Vigo D, editors. The vehicle routing problem. Philadelphia: SIAM; 2001. p. 29–52.
[4] Naddef D, Rinaldi G. Branch-and-cut algorithms for the capacitated VRP. In: Toth P, Vigo D, editors. The vehicle routing problem. Philadelphia: SIAM; 2001. p. 53–84.

[5] Bramel J, Simchi-Levi J. Set-covering-based algorithms for the capacitated VRP. In: Toth P, Vigo D, editors. The vehicle routing problem. Philadelphia: SIAM; 2001. p. 85–108.

[6] Clarke G, Wright JW. Scheduling of vehicles from a central depot to a number of delivery points. Operations Research 1964;12:568–81.

[7] Gillett BE, Miller LR. A heuristic algorithm for the vehicle dispatch problem. Operations Research 1974;22:340–9.

[8] Christofides N, Mingozzi A, Toth P. The vehicle routing problem. In: Christofides N, Mingozzi A, Toth P, Sandi C, editors. Combinatorial optimization. Chichester: Wiley; 1979. p. 315–38.

[9] Paessens H. The savings algorithm for the vehicle routing problem. European Journal of Operational Research 1988;34:336–44.

[10] Renaud J, Boctor FF, Laporte G. An improved petal heuristic for the vehicle routing problem. Journal of the Operational Research Society 1996;47:329–36.

[11] Laporte G, Gendreau M, Potvin J-Y, Semet F. Classical and modern heuristics for the vehicle routing problem. International Transactions in Operations Research 2000;7:285–300.

[12] Laporte G, Semet F. Classical heuristics for the capacitated VRP. In: Toth P, Vigo D, editors. The vehicle routing problem. Philadelphia: SIAM; 2001. p. 109–28.

[13] Van Breedam A. A parametric analysis of heuristics for the vehicle routing problems with side constraints. European Journal of Operational Research 2002;137:348–70.

[14] Lin S. Computer solutions of the traveling salesman problem. Bell System Technical Journal 1965;44:2245–69.

[15] Or I. Traveling salesman-type combinatorial problems and their relation to the logistics of blood banking. PhD thesis, Northwestern University, Evanston, USA; 1976.

[16] Thompson PM, Psaraftis HN. Cyclic transfer algorithms for multi-vehicle routing and scheduling problems. Operations Research 1993;41: 935–46.

[17] Van Breedam A. An analysis of the behavior of heuristics for the vehicle routing problem for a selection of problems with vehicle-related, customer-related, and time-related constraints. PhD dissertation, University of Antwerp, Belgium; 1994.

[18] Kindervater GAP, Savelsbergh MWP. Vehicle routing: handling edge exchanges. In: Aarts EHL, Lenstra JK, editors. Local search in combinatorial optimization. Chichester: Wiley; 1997. p. 337–60.

[19] Ergun Ö, Orlin JB, Steele-Feldman A. Creating very large scale neighborhoods out of smaller ones by compounding moves: a study on the vehicle routing problem. Working paper, Georgia Institute of Technology, USA; 2004.

[20] Cordeau J-F, Gendreau M, Laporte G, Potvin J-Y, Semet F. A guide to vehicle routing heuristics. Journal of the Operational Research Society 2002;53:512–22.

[21] Osman IH. Metastrategy simulated annealing and tabu search algorithms for the vehicle routing problems. Annals of Operations Research 1993;41:421–52.

[22] Taillard E. Parallel iterative search methods for vehicle routing problems. Networks 1993;23:661–73.

[23] Gendreau M, Hertz A, Laporte G. A tabu search heuristic for the vehicle routing problem. Management Science 1994;40:1276–90.

[24] Rochat Y, Taillard E. Probabilistic diversification and intensification in local search for vehicle routing. Journal of Heuristics 1995;1:147–67.

[25] Xu J, Kelly JP. A network flow-based tabu search heuristic for the vehicle routing problem. Transportation Science 1996;30:379–93.

[26] Barbarosoglu G, Ozgur D. A tabu search algorithm for the vehicle routing problem. Computers & Operations Research 1999;26:255–70.

[27] Cordeau J-F, Laporte G, Mercier A. A unified tabu search heuristic for vehicle routing problems with time windows. Journal of the Operational Research Society 2001;52:928–36.

[28] Toth P, Vigo D. The granular tabu search (and its application to the vehicle routing problem). INFORMS Journal on Computing 2003;15: 333–48.

[29] Tarantilis CD, Kiranoudis CT, Vassiliadis VS. A backtracking adaptive threshold accepting metaheuristic for the vehicle routing problem. Journal of Systems Analysis Modelling Simulation 2002;42:631–44.

[30] Tarantilis CD, Kiranoudis CT, Vassiliadis VS. A list based threshold accepting algorithm for the capacitated vehicle routing problem. International Journal on Computer Mathematics 2002;79:537–53.

[31] Li F, Golden B, Wasil E. Very large-scale vehicle routing: new test problems, algorithms, and results. Computers & Operations Research 2005;32:1165–79.

[32] Berger J, Barkaoui M. Hybrid genetic algorithm for the capacitated vehicle routing problem. In: Proceedings of the genetic and evolutionary computation conference, Chicago; 2003. p. 646–56.

[33] Prins C. A simple and effective evolutionary algorithm for the vehicle routing problem. Computers & Operations Research 2004;31: 1985–2002.

[34] Baker BM, Ayechew MA. A genetic algorithm for the vehicle routing problem. Computers & Operations Research 2003;30:787–800.

[35] Mester D, Bräysy O. Active guided evolution strategies for large scale vehicle routing problem. Working paper, University of Haifa, Israel; 2004.

[36] Tarantilis CD, Kiranoudis CT. Boneroute: an adaptive memory-based method for effective fleet management. Annals of Operations Research 2002;115:227–41.

[37] Tarantilis CD. Solving the vehicle routing problem with adaptive memory programming methodology. Computers & Operations Research 2005;32:2309–27.

[38] Bullnheimer B, Hartl RF, Strauss C. Applying the ant system to the vehicle routing problem. In: Voss S, Martello S, Osman IH, Roucairol C, editors. Meta-heuristics: advances and trends in local search paradigms for optimization. Boston: Kluwer; 1998. p. 109–20.

[39] Bullnheimer B, Hartl RF, Strauss C. An improved ant system for the vehicle routing problem. Annals of Operations Research 1999;89:319–28.

[40] Reimann M, Doerner K, Hartl RF. D-ants: savings based ants divide and conquer the vehicle routing problem. Computers & Operations Research 2004;31:563–91.

[41] Baker BM, Carreto CAC. A visual interactive approach to vehicle routing. Computers & Operations Research 2003;30:321–37.

[42] Van Breedam A. Comparing descent heuristics and metaheuristics for the vehicle routing problem. Computers & Operations Research 2001;28:289–315.
[43] Gendreau M, Laporte G, Potvin J-Y. Metaheuristics for the capacitated VRP. In: Toth P, Vigo D, editors. The vehicle routing problem. Philadelphia: SIAM; 2001. p. 129–54.
[44] Cordeau J-F, Gendreau M, Hertz A, Laporte G, Sormany, JS. New heuristics for the vehicle routing problem. In: Langevin A, Riopel D, editors. Logistics systems: design and optimization. Boston: Kluwer.
[45] Golden BL, Wasil EA, Kelly JP, Chao I-M. The impact of metaheuristics on solving the vehicle routing problem: algorithms, problem sets, and computational results. In: Crainic T, Laporte G, editors. Fleet management and logistics. Boston: Kluwer; 1998. p. 33–56.
[46] Assad AA, Golden BL. Arc routing methods and applications, In: Ball MO, Magnanti TL, Monma CL, Nemhauser GL, editors. Handbooks in operations research and management science 8: network routing. Amsterdam: North-Holland. p. 375–484.
[47] Mladenovic N, Hansen P. Variable neighbourhood search. Computers & Operations Research 1997;24:1097–100.
[48] Voudouris C. Guided local search for combinatorial problems. PhD dissertation, University of Essex, UK; 1997.
[49] Voudouris C, Tsang E. Guided local search and its application to the TSP. European Journal of Operations Research 1998;113:80–119.
[50] Flood MM. The traveling-salesman problem. Operations Research 1956;4:61–75.
[51] Savelsbergh MWP. The vehicle routing problem with time windows: minimizing route duration. INFORMS Journal on Computing 1992;4: 146–54.
[52] Potvin J-Y, Rousseau J-M. An exchange heuristic for routing problems with time windows. Journal of the Operational Research Society 1995;46:1433–46.
[53] Taillard E, Badeau P, Gendreau M, Guertin F, Potvin J-Y. A tabu search heuristic for the vehicle routing problem with soft time windows. Transportation Science 1997;31:170–86.
[54] Dueck G, Scheurer T. Threshold accepting: a general purpose optimization algorithm appearing superior to simulated annealing. Journal of Computational Physics 1990;90:161–75.
[55] Knuth DE. The art of computer programming, vol. 2. Boston: Addison-Wesley; 1998.
[56] Cornea-Hasegan M, Norin B. IA-64 Floating-point operations and the IEEE standard for binary floating-point arithmetic. Intel Technology Journal 1999;4.
[57] Markstein P. Software division and square root using Goldschmidt's algorithms. Real Numbers and Computers 2004;6:146–57.
[58] Zhuang X, Lee HHS. A hardware-based cache pollution filtering mechanism for aggressive prefetches. In: Proceedings of the 2003 international symposium on parallel processing ICPP'03, October 2003, Kaohsiung, Taiwan. p. 286–93.
[59] Kytöjoki J. New implementation techniques for search algorithms. Working paper, University of Jyväskylä, Finland; 2004.