Computers & Operations Research 36 (2009) 1639 – 1645

Contents lists available at ScienceDirect

Computers & Operations Research

journal homepage: www.elsevier.com/locate/cor

A hybrid GRASP/VND heuristic for the one-commodity pickup-and-delivery traveling salesman problem

Hipólito Hernández-Pérez, Inmaculada Rodríguez-Martín, Juan José Salazar-González*

DEIOC, Facultad de Matemáticas, Universidad de La Laguna, 38271 La Laguna, Tenerife, Spain

ARTICLE INFO

Available online 29 March 2008

Keywords:
Pickup and delivery
Traveling salesman problem
Hybrid heuristic
GRASP
VND

ABSTRACT

We address in this paper the one-commodity pickup-and-delivery traveling salesman problem, which is characterized by a set of customers, each of them supplying (pickup customer) or demanding (delivery customer) a given amount of a single product. The objective is to design a minimum cost Hamiltonian route for a capacitated vehicle in order to transport the product from the pickup to the delivery customers. The vehicle starts the route from a depot, and its initial load also has to be determined. We propose a hybrid algorithm that combines the GRASP and VND metaheuristics. Our heuristic is compared with other approximate algorithms described in Hernández-Pérez and Salazar-González [Heuristics for the one-commodity pickup-and-delivery traveling salesman problem. Transportation Science 2004;38:245–55]. Computational experiments on benchmark instances reveal that our hybrid method yields better results than the previously proposed approaches.

1. Introduction

The *one-commodity pickup-and-delivery traveling salesman problem* (1-PDTSP) is a routing problem that generalizes the classical *traveling salesman problem* (TSP). We are given a set of locations and the travel distances among them. One specific location is considered to be a vehicle *depot*, while all the others are identified with *customers*. There is a unique commodity or product that has to be transported from some customers to others. To this end, each customer is visited once by the vehicle. Customers are divided into two groups, depending on whether they supply a given amount of product (*pickup customers*) or they demand a given amount of it (*delivery customers*). The product collected at pickup customers can be supplied to delivery customers. Moreover, the vehicle has a known capacity, and it must start and end its route at the depot. The 1-PDTSP consists of finding a minimum length Hamiltonian route for the vehicle that satisfies all the customer requirements. It is not assumed that the vehicle leaves empty or full loaded from the depot. On the contrary, the initial load of the vehicle also has to be determined (see [1]). We will assume that the travel distances among locations are symmetric.

The 1-PDTSP has several practical applications in routing a single commodity. One of them is described by Anily and Bramel [2] in the context of inventory repositioning. Consider a set retailers owned by the same firm and located at different sites in a region. At a given moment, due to the random nature of demands, some retailers may have an excess of inventory while others are in need of additional stock. Then, the firm may decide to transfer inventory from the first group of retailers to the second one. Determining the cheapest Hamiltonian route to do so with a capacitated vehicle is exactly the 1-PDTSP. Anily and Bramel [2] propose heuristic algorithms for the special case of the 1-PDTSP where the delivery and pickup quantities are all equal to one unit. This problem is called *capacitated TSP with pickups and deliveries*. Chalasani and Motwani [3] consider the same problem with the name *Q-delivery TSP*. The problem with unitary pickups and deliveries on a path or tree network has been studied by Wang et al. [4].

Hernández-Pérez [5] is the first to introduce the 1-PDTSP. He makes a theoretical study of the problem and presents solution methods. Hernández-Pérez and Salazar-González [6] describe an exact branch-and-cut algorithm able to solve instances with up to 60 customers. The same authors propose in [1] two heuristic approaches to deal with larger instances. The first heuristic approach is a simple local search procedure developed to provide initial upper bounds for their branch-and-cut algorithm. The second approach is a more elaborated algorithm based on "incomplete optimization". That is, the branch-and-cut algorithm described in [6] is applied to a restricted search space obtained by considering only a subset of model variables, associated to promising edges of a graph. Moreover, the branch-and-cut execution is truncated by imposing a limit to the number of levels in the decision tree exploration (see [1] for details). A primal heuristic is also embedded in the branch-and-cut to build feasible integer solutions from the information given by the

* Corresponding author. Tel.: +34 922 318184; fax: +34 922 318170.
E-mail address: jjsalaza@ull.es (J.J. Salazar-González).

0305-0548/$ - see front matter © 2008 Elsevier Ltd. All rights reserved.
doi:10.1016/j.cor.2008.03.008

fractional solutions. This procedure is periodically applied during the search process.

There are many other pickup-and-delivery routing problems described in the literature. For recent surveys, we refer the reader to Savelsbergh and Sol [7], Parragh et al. [8,9], and Berbeglia et al. [10]. However, as observed in [9], little attention has been paid to the 1-PDTSP. As far as we know, the only heuristic approaches are those in [1], none of these is a metaheuristic.

The 1-PDTSP is $\mathcal{NP}$-hard since it coincides with the TSP when the vehicle capacity is large enough. Even more, the problem of checking the existence of a feasible solution is $\mathcal{NP}$-complete in the strong sense (see [5]). This is a fundamental difference with respect to the TSP, as even just finding a feasible tour may be a very complex task. In this article we present a hybrid heuristic method that combines a greedy randomized adaptive search procedure (GRASP) with variable neighborhood descent (VND). The proposed algorithm is compared with the heuristic methods described in [1]. The outcomes of the computational tests show that the new heuristic yields better results than the previous ones, managing to improve the best known solution for most large instances.

We introduce now the notation used throughout this article. The depot is denoted by 1 and each customer by i ($i = 2, \ldots, n$). The set $V := \{1, 2, \ldots, n\}$ is the vertex set and E is the edge set. For each pair of locations $\{i, j\}$, the travel distance (or cost) c_{ij} of traveling between i and j is given. A non-zero demand q_i associated with each customer i is also given, being $q_i < 0$ if i is a delivery customer and $q_i > 0$ if i is a pickup customer. The capacity of the vehicle is represented by Q and is assumed to be a positive number. Note that typically $Q \leqslant \max\{\sum_{i \in V: q_i > 0} q_i, -\sum_{i \in V: q_i < 0} q_i\}$ on a 1-PDTSP instance. The depot can be considered a customer by defining $q_1 := -\sum_{i=2}^{n} q_i$, i.e., a customer absorbing or providing the necessary amount of product to ensure product conservation.

The remainder of this article is organized as follows. Section 2 describes our algorithm and its constituent parts. The computational results in Section 3 show the effectiveness of our method, that improves the performance of the heuristics presented in [1]. Final remarks are made in Section 4.

2. The algorithm

Search based heuristics for combinatorial optimization problems usually require some kind of diversification to overcome local optimality. Multi-start methods seek diversification by re-starting a local search procedure from multiple randomly generated initial solutions. The GRASP metaheuristic, proposed by Feo and Resende [11], is a multi-start procedure. Therefore it consists basically of a loop embedding a construction phase and a local search phase. The best overall solution is kept as the final result. The construction phase builds up a solution iteratively, randomly selecting each time an element from a *restricted candidate list* (RCL). The elements in the list are sorted according to a greedy function previously defined. This function measures the benefit of selecting each element. The procedure is adaptive since the benefits associated to every element are updated at each iteration of the construction phase, reflecting the changes brought on by the selection of the previous elements. The probabilistic component of a GRASP is characterized by a random choice of the element from the list, that is not necessarily the top candidate of the RCL. This choice technique allows for different solutions to be generated at each GRASP iteration. The whole strategy has been successfully applied to solve several difficult optimization problems (see Festa and Resende [12] for a review, and Resende and Ribeiro [13]).

On the other hand, VNS (variable neighborhood search) is based on the systematic change of neighborhood within the search (see Mladenović and Hansen [14]). The key idea is to change the local search operator, or neighborhood, once a local optimum is attained. To rapidly expose the main steps of VNS, let us denote by N_k ($k = 1, \ldots, k_{\max}$) a set of pre-selected neighborhood structures, by x a given solution, and by $N_k(x)$ the set of neighbor solutions of x in the k-th neighborhood. The algorithm performs a series of iterations until a stopping condition is satisfied. At each iteration, and starting with $k = 1$, a neighbor solution $x' \in N_k(x)$ is randomly generated. Then, a local search is applied to x' producing a local optimum. If the local optimum improves the current solution, then x is updated and the process is repeated. Otherwise, the algorithm resumes from x using a higher order neighborhood, if there is any. The VND method is a variant of VNS (see [14]) where the change of neighborhood is performed in a deterministic way. More precisely, the local minimum found when performing local search within a neighborhood is the starting point of the local search within the next neighborhood. The basic scheme of VND is stated in Algorithm 1.

Algorithm 1. VND(x) procedure
  for $k \leftarrow 1$ to *max* **do**
    $x' \leftarrow$ LocalSearch($x, N_k(x)$)
    if x' is better than x **then**
      $x \leftarrow x'$
    end if
  end for
  return x

The algorithm we propose for solving the 1-PDTSP is a hybrid algorithm that combines the GRASP and the VND paradigms. The first part consists of a GRASP where the local search has been replaced by a VND procedure. That is, at each iteration of the GRASP loop, the solution given by the greedy randomized algorithm is taken as the starting point of a first VND, referred to as VND_1. This procedure is composed of two edge-exchange neighborhood structures. The GRASP loop is iterated until a certain stopping condition is met. Then, it follows the second part of the heuristic, a post-optimization phase consisting of a second VND, called VND_2, that starts from the best solution found so far. The procedure VND_2 uses two neighborhood structures based on vertex-exchange movements. The whole scheme of the hybrid heuristic is outlined in Algorithm 2.

Algorithm 2. Hybrid heuristic for the 1-PDTSP
 while stopping criterion is not satisfied **do**
  $x \leftarrow$ GreedyRandomizedInitSol() {`construction phase`}
  {`improvement phase`}
  $x \leftarrow$ VND_1(x) {`edge-exchange neighborhoods`}
  if x is feasible and improves the best solution x' **then**
   $x' \leftarrow x$
  end if
 end while
 {`post-optimization`}
 $x' \leftarrow$ VND_2(x') {`vertex-exchange neighborhoods`}
 return x'

Next we describe with more detail each of the hybrid heuristic components.

2.1. Construction phase

To generate an initial solution we proceed in a greedy and adaptive way, starting from a randomly selected customer and iteratively adding a new one each time until all customers are in the solution. Recall that a partial solution corresponds to a path for the vehicle from the first to the last customer in the solution, and that, as explained in [1], it is feasible only if the difference between the maximum and the minimum load of the vehicle along the path does not exceed the capacity. More precisely, let $\vec{P}$ be a path through the sequence of customers $i_1, \ldots, i_k$, with $k \leqslant n$. Let $l_0(\overrightarrow{P}) := 0$, and let $l_j(\overrightarrow{P}) := l_{j-1}(\overrightarrow{P}) + q_{i_j}$ be the load of the vehicle when leaving customer i_j, with $j = 1, \ldots, k$. Note that $l_j(\overrightarrow{P})$ may be negative. Then, $\overrightarrow{P}$ is *feasible* only if

$$infeas(\overrightarrow{P}) := \max_{j=0,\ldots,k}\{l_j(\overrightarrow{P})\} - \min_{j=0,\ldots,k}\{l_j(\overrightarrow{P})\} - Q \leqslant 0.$$

Therefore, evaluating the feasibility of a partial solution is a linear-time task.

At each iteration a customer is added at the end of the path, trying to obtain a feasible solution of good quality, i.e., with small length. To this end, we evaluate all possible extensions of the actual solution, and retain only feasible candidates. Next we sort them according to their distance to the last customer in the path, and include the first l in a RCL, l being the minimum between 10 and the number of feasible candidates. If there are no feasible candidates at all, then the RCL is filled with the l nearest customers to the last one in the path, now l being the minimum between 10 and the number of customers not in the solution. Finally, one customer from the RCL is chosen randomly and it is added to the solution under construction, thus extending the path.

The travel distances used in this construction phase are redefined as was done in [1], in order to slightly penalize the edges connecting customers of the same type (i.e., pickup and pickup, or delivery and delivery). The idea is to favor the inclusion in the tour under construction of edges joining customers of different type, since this increases the probability of finding a feasible solution for the 1-PDTSP (see [1] for more explanations). However, there is no guarantee that the procedure ends up with a feasible solution.

2.2. Improvement phase

The improvement phase is a VND algorithm (called VND_1 in Algorithm 2) that systematically applies two tour improvement procedures to each initial solution provided by the construction phase. The two local search operators are adaptations of the 2-opt and 3-opt edge-exchange operators described in Lin [15] and Johnson and McGeoch [16], for the TSP. Following the ideas in Lin and Kernighan [17], for each vertex or customer i we store a list of its nearest neighbors j, sorted in increasing order of c_{ij}, and only edges $\{i,j\}$ with j in the list are considered for inclusion in the tour when performing the exchanges. This rule substantially reduces the search effort, and therefore the computing time. The size of the lists of neighbors is related to the number n of customers, and it is set to $4\sqrt{n}$.

On the other hand, as we mentioned before, the construction phase may end up with an infeasible solution, that is, with a Hamiltonian route that violates the capacity constraint for the vehicle. Consequently, VND_1 has to be able to handle infeasible solutions. If only feasible movements are accepted, the local search can easily get stuck in a local optimum. Therefore, we have also modified the classical 2-opt and 3-opt procedures so as to guide the local searches towards feasible solutions, besides reducing the cost. In order to do so, we define an accepting threshold t for the infeasibility. Edge-exchange movements leading to a solution S with a smaller cost are accepted only if S is feasible or if its infeasibility measure is under the threshold, i.e., $infeas(S) < t$. If the new solution S is feasible, then the threshold is set to $t := \varepsilon$ (being $\varepsilon > 0$ a very small value) and it is not further changed, thus forbidding infeasible movements from that moment on. Otherwise, the threshold is updated to $t := infeas(S)$. In this way, the allowed infeasibility bound is progressively reduced during the search. The initial value of the bound is set to $t := 3\max\{\sum_{i \in V: q_i > 0} q_i, -\sum_{i \in V: q_i < 0} q_i\}/n$.

Note that, if the solution acting as starting point of the 2-opt or 3-opt procedures is already feasible, the described mechanism allows, at the beginning of the local search, movements leading to less costly but infeasible solutions, although these solutions are soon reconducted to feasibility. In this sense, our handling of the infeasibility threshold works not only as a tool for leading the search towards feasibility, but also as a tool for escaping from local optima.

Usually in a VND procedure, the sizes of the neighborhoods and the time complexities of their evaluation procedures induce a natural order for them within the whole scheme, so that smaller or faster neighborhoods are examined first. We follow this rule within VND_1, and perform first 2-opt and then 3-opt. Inside any of them, we opt for making the best edge-exchange movement each time, instead of just making the first improving movement.

2.3. Stopping criterion

The loop embedding the construction phase and the improvement phase is repeated until a stopping condition is met. Recall that the number of iterations of the loop is related with both running time and solution quality. In a typical GRASP implementation, the loop is repeated a given number of times. In our case, the stopping criterion combines a limit to the total number of iterations (set to 200), and a limit to computation time (set to 600 s). The limit on computation time is only relevant for the hardest instances, with a large number of customers and a small vehicle capacity, where the local search routines become very time consuming.

2.4. Post-optimization

Once the GRASP loop is over, a post-optimization procedure is applied to the best solution found so far, hoping to further improving it. The post-optimization procedure is a VND algorithm (called VND_2 in Algorithm 2) with two neighborhood structures defined by two different vertex-exchange operators. The first one, that we call *move forward operator*, tries to find a better solution, i.e., a shorter tour, by moving a customer from its current position i in the tour, to some further position j with $j > i$. This implies that all customers in positions $i+1, \ldots, j$ have to be shifted backwards one position. Customers in positions 1 to $i-1$ and $j+1$ to n remain unchanged. See Fig. 1.

For a given customer at position i, the algorithm checks for reallocation in all possible positions $j = i+1, \ldots, n$. Movements leading to infeasible solutions are discarded and, in order to save time, the first improving exchange is made, instead of looking for the best movement. The local search continues until no further improvement is obtained.

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

Both neighborhood structures have the same size and their exploration requires an equivalent computational effort. Therefore, its order within VNS_2 does not seem to be relevant. We decided to perform first the move forward, and then the move backward local search.

Finally, let us comment that in the initial implementations of the hybrid method, these two neighborhood structures were part of VND_1, that is, the corresponding local searches were performed within the GRASP loop and there was no post-optimization phase. This resulted in an increase of the solution quality, but at the cost of also augmenting considerably the computing time. We finally opted for the flowchart in Algorithm 2 as a tradeoff between both factors. For the same reason, some other neighborhoods based on swap and exchange operators, that were implemented and tested, do not appear in the final algorithm.

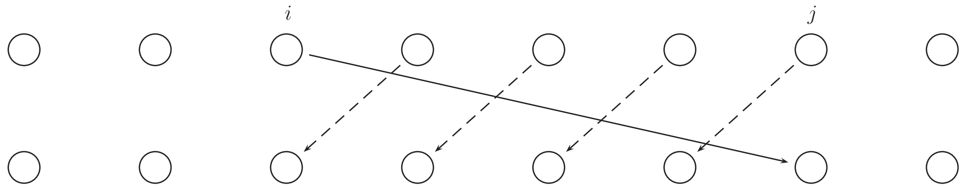

Fig. 1. Move forward operator: customer in position i is moved ahead to position j.

Table 1
Heuristic results for small instances.

n	Name	$Q=10$				$Q=20$				$Q=40$			
		OPT	H-best	H-aver.	Time	OPT	H-best	H-aver.	Time	OPT	H-best	H-aver.	Time
20	A	4963	**4963**	4963.0	0.07	3816	**3816**	3816.0	0.02	3816	**3816**	3816.0	0.01
	B	4976	**4976**	4976.0	0.06	4224	**4224**	4224.0	0.02	3942	**3942**	3942.0	0.02
	C	6333	**6333**	6333.0	0.10	4492	**4492**	4492.0	0.04	3897	**3897**	3897.0	0.01
	D	6280	**6280**	6280.0	0.07	4706	**4706**	4706.0	0.04	3743	**3743**	3743.0	0.01
	E	6415	**6415**	6415.0	0.07	4673	**4673**	4673.0	0.03	4299	**4299**	4299.0	0.02
	F	4805	**4805**	4805.0	0.08	4118	**4118**	4118.0	0.02	4118	**4118**	4118.0	0.02
	G	5119	**5119**	5119.0	0.04	4369	**4369**	4369.0	0.02	4248	**4248**	4248.0	0.02
	H	5594	**5594**	5594.0	0.06	4159	**4159**	4159.0	0.02	4007	**4007**	4007.0	0.01
	I	5130	**5130**	5130.0	0.10	4116	**4116**	4116.0	0.02	4026	**4026**	4026.0	0.02
	J	4410	**4410**	4410.0	0.08	3700	**3700**	3700.0	0.02	3678	**3678**	3678.0	0.02
30	A	6403	**6403**	6406.8	0.43	4918	**4918**	4918.0	0.08	4620	**4620**	4620.0	0.05
	B	6603	**6603**	6603.0	0.24	5109	**5109**	5109.0	0.09	4529	**4529**	4529.0	0.05
	C	6486	**6486**	6486.0	0.21	4901	**4901**	4901.0	0.08	4377	**4377**	4377.0	0.05
	D	6652	**6652**	6655.1	0.40	5385	**5385**	5385.0	0.08	4876	**4876**	4876.0	0.05
	E	6070	**6070**	6070.0	0.39	4916	**4916**	4916.0	0.06	4822	**4822**	4822.0	0.05
	F	5737	**5737**	5737.0	0.37	4459	**4459**	4459.0	0.06	4390	**4390**	4390.0	0.05
	G	9371	**9371**	9371.0	0.30	6672	**6672**	6672.7	0.16	5048	**5048**	5048.0	0.06
	H	6431	**6431**	6431.2	0.33	4684	**4684**	4684.0	0.07	4583	**4583**	4583.0	0.05
	I	5821	**5821**	5821.0	0.25	4483	**4483**	4483.0	0.05	4379	**4379**	4379.0	0.05
	J	6187	**6187**	6187.4	0.38	4645	**4645**	4645.0	0.06	4421	**4421**	4421.0	0.05
40	A	7173	**7173**	7188.5	0.67	5481	**5481**	5481.0	0.14	5124	**5124**	5124.0	0.09
	B	6557	**6557**	6568.5	0.91	5334	**5334**	5334.0	0.11	5315	**5315**	5315.0	0.09
	C	7528	**7528**	7528.4	0.66	5775	**5775**	5775.0	0.22	4916	**4916**	4916.0	0.09
	D	8059	**8059**	8135.6	1.00	6054	**6054**	6056.6	0.19	5538	**5538**	5538.0	0.09
	E	6928	**6928**	6959.3	1.04	5598	**5598**	5598.0	0.14	5364	**5364**	5364.0	0.09
	F	7506	**7506**	7590.5	0.83	5491	**5491**	5491.0	0.15	5059	**5059**	5059.0	0.09
	G	7624	**7624**	7682.8	0.75	5588	**5588**	5588.0	0.14	5366	**5366**	5366.0	0.09
	H	6791	**6791**	6795.7	0.83	5141	**5141**	5141.0	0.14	4837	**4837**	4837.0	0.09
	I	7215	**7215**	7219.0	0.76	5262	**5262**	5262.0	0.14	4967	**4967**	4967.0	0.09
	J	6512	**6512**	6513.3	0.53	5277	**5277**	5277.0	0.13	4939	**4939**	4939.0	0.09
50	A	6987	**6987**	6996.7	0.96	5908	**5908**	5908.0	0.18	5816	**5816**	5816.0	0.14
	B	9488	**9488**	9512.6	1.73	7111	**7111**	7150.1	0.48	6249	**6249**	6251.1	0.20
	C	9110	**9110**	9133.7	1.76	6962	**6962**	6965.0	0.46	6284	**6284**	6284.0	0.19
	D	10 260	**10 260**	10 464.3	1.82	7278	**7278**	7285.2	0.53	6423	**6423**	6423.0	0.20
	E	9492	**9492**	9625.1	1.85	7107	**7107**	7133.0	0.42	6224	**6224**	6224.0	0.19
	F	8684	**8684**	8773.2	1.72	6053	**6053**	6068.5	0.28	5453	**5453**	5453.0	0.15
	G	7126	**7126**	7217.4	1.34	5968	**5968**	5968.0	0.20	5881	**5881**	5881.0	0.16
	H	8885	8895	9006.5	1.46	6477	**6477**	6491.8	0.33	5642	**5642**	5642.0	0.16
	I	8329	**8329**	8412.5	0.89	6149	**6149**	6149.0	0.26	5572	**5572**	5572.0	0.15
	J	8456	**8456**	8666.1	1.61	6364	**6364**	6364.0	0.28	5915	**5915**	5915.0	0.15
60	A	8602	**8602**	8726.6	2.37	6696	**6696**	6726.0	0.43	6156	**6156**	6156.0	0.24
	B	8514	**8514**	8683.2	2.38	6730	**6730**	6734.3	0.41	6524	**6524**	6524.0	0.23
	C	9453	**9453**	9565.6	2.70	7081	**7081**	7081.5	0.56	6240	**6240**	6240.0	0.27
	D	11 059	11 061	11 320.6	2.62	8011	8064	8119.0	0.75	6855	**6855**	6855.0	0.30
	E	9487	9572	9724.8	2.56	7317	**7317**	7342.3	0.52	6556	**6556**	6556.0	0.28
	F	9063	**9063**	9437.2	2.36	6449	**6449**	6473.9	0.43	6154	**6154**	6154.0	0.24
	G	8912	8967	9107.9	2.49	6882	**6882**	6882.0	0.48	6322	**6322**	6322.0	0.27
	H	8424	**8424**	8467.3	2.19	6444	**6444**	6449.9	0.38	6087	**6087**	6087.0	0.24
	I	9394	**9394**	9529.6	1.99	6933	**6933**	6949.0	0.58	6072	**6072**	6072.0	0.25
	J	8750	**8750**	8956.5	2.29	7017	**7017**	7041.5	0.36	6651	**6651**	6651.0	0.23

3. Computational results

The algorithm was implemented in C++ and the program was run on a personal computer with Intel Core 2 CPU at 2.4 GHz under Windows XP. Computational experiments were carried on the benchmark instances used in [1,6]. These instances were obtained with a random generator similar to the one proposed in Mosheiov [18], as follows. For each value of n, $n-1$ customers were generated with random coordinates in the square $[-500, 500] \times [-500, 500]$, and with a random demand in $[-10, 10]$. The depot is located at $(0, 0)$ and has demand $q_1 := -\sum_{i=2}^{n} q_i$. The travel cost c_{ij} was computed as the Euclidean distance between i and j. The instances are characterized by the number of customers n and the vehicle capacity Q, and they can be divided in two classes:

Table 2
Heuristic results for large instances.

n	Name	$Q=10$				$Q=20$				$Q=40$			
		*UB*2	H-best	H-aver.	Time	*UB*2	H-best	H-aver.	Time	*UB*2	H-best	H-aver.	Time
100	A	12 042	**11 874**	12 087.6	8.48	8768	**8616**	8779.2	1.63	**7938**	**7938**	7941.8	0.60
	B	**13 172**	13 288	13 582.6	10.23	9629	**9536**	9686.9	2.51	8144	**8124**	8182.6	0.72
	C	**14 063**	14 069	14 421.3	10.27	10 099	**9993**	10 191.2	3.03	**8441**	**8441**	8514.5	0.81
	D	**14 490**	14 542	14 787.5	8.95	10 464	**10 064**	10 340.7	3.07	8380	**8264**	8360.0	0.82
	E	**11 546**	11 650	12 502.6	6.13	8929	**8838**	8986.2	1.42	**7960**	**7960**	7996.5	0.58
	F	12 021	**11 734**	12 010.7	7.67	9056	**9029**	9106.2	1.63	**8074**	**8074**	8116.1	0.56
	G	12 170	**12 049**	12 366.9	7.82	9022	**8865**	9078.5	1.58	8183	**8168**	8189.0	0.60
	H	13 056	**12 892**	13 169.2	9.39	9708	**9495**	9681.0	2.36	**7992**	**7992**	8022.3	0.74
	I	14 191	**14 048**	14 390.2	7.94	10 144	**10 005**	10 192.7	2.41	8484	**8440**	8504.1	0.71
	J	13 439	**13 430**	13 737.6	11.65	9835	**9742**	9922.7	2.56	**8255**	**8255**	8289.1	0.71
200	A	**18 013**	18 145	18 564.0	36.00	13 455	**13 422**	13 714.8	11.01	**11 136**	11 156	11 369.3	3.42
	B	**18 154**	18 520	18 932.5	33.68	**13 242**	13 419	13 714.8	12.32	11 305	**11 296**	11 489.6	3.48
	C	17 305	**16 969**	17 280.3	41.01	**12 264**	12 314	12 678.5	8.93	10 919	**10 849**	11 038.4	3.01
	D	**21 565**	21 848	22 285.7	33.51	15 387	**15 212**	15 548.6	24.59	12 002	**11 802**	12 037.4	5.39
	E	20 033	**19 913**	20 643.2	39.75	14 109	**14 066**	14 298.3	18.41	11 276	**11 237**	11 474.7	4.70
	F	22 090	**21 949**	22 284.6	80.93	**15 105**	15 167	15 542.0	30.87	11 931	**11 836**	11 988.1	6.35
	G	**17 956**	18 035	18 627.7	28.58	13 203	**13 200**	13 495.7	11.43	11 174	**11 154**	11 302.0	3.54
	H	21 995	**21 463**	22 084.9	47.45	15 518	**15 278**	15 571.8	26.74	12 234	**12 088**	12 430.2	5.87
	I	18 695	**18 606**	19 184.8	34.31	**13 082**	13 338	13 597.8	13.63	11 272	**11 115**	11 298.1	3.66
	J	19 349	**19 273**	19 839.5	42.43	14 043	**13 870**	14 159.4	15.80	11 181	**11 123**	11 281.5	4.07
300	A	**23 244**	23 566	24 052.9	112.51	**16 830**	16 920	17 242.8	41.74	**13 670**	13 787	14 008.1	11.32
	B	23 256	**23 187**	23 845.6	109.55	**16 844**	17 050	17 248.4	39.31	13 881	**13 875**	14 127.2	10.95
	C	22 276	**21 800**	22 516.6	104.58	16 548	**16 364**	16 661.1	34.11	**13 489**	13 642	13 792.7	9.79
	D	26 434	**25 971**	26 462.1	162.95	**18 024**	18 178	18 651.7	61.75	14 477	**14 426**	14 733.6	14.66
	E	27 931	**27 420**	27 892.1	139.56	19 130	**18 715**	19 088.5	86.47	14 616	**14 521**	14 902.1	20.21
	F	25 096	**24 852**	25 278.2	153.93	18 216	**18 126**	18 387.1	63.61	14 390	**14 345**	14 665.7	15.30
	G	24 363	**24 308**	24 760.5	151.22	17 490	**17 363**	17 759.4	53.70	14 299	**14 151**	14 382.6	13.10
	H	22 869	**22 684**	23 116.5	67.49	16 759	**16 725**	16 997.7	32.67	13 816	**13 674**	14 047.1	9.39
	I	25 157	**24 633**	25 492.6	76.72	18 048	**17 654**	17 996.0	60.08	14 396	**14 232**	14 489.2	14.58
	J	23 468	**23 086**	23 530.2	100.05	17 027	**16 811**	17 168.5	35.51	**13 759**	13 963	14 127.7	9.58
400	A	31 821	**31 486**	31 912.0	282.00	21 741	**21 617**	22 042.2	198.75	16 966	**16 939**	17 198.2	41.83
	B	**24 883**	25 243	25 606.4	204.21	**18 459**	19 021	19 260.7	60.09	16 027	**16 013**	16 217.0	18.56
	C	29 044	**28 942**	29 463.2	246.29	20 827	**20 765**	21 172.0	125.79	**16 506**	16 588	16 964.1	31.03
	D	24 639	**24 597**	25 308.6	142.84	18 443	**18 375**	18 767.0	49.26	**15 691**	15 801	16 033.2	15.39
	E	**25 548**	25 644	26 120.0	219.87	**18 598**	18 764	19 153.6	64.63	15 658	**15 638**	15 906.5	18.91
	F	27 215	**27 169**	27 755.1	273.01	20 112	**19 941**	20 223.8	85.24	**16 085**	16 373	16 541.0	23.52
	G	24 728	**24 626**	25 088.4	181.55	18 695	**18 624**	18 900.5	55.50	**15 603**	15 716	15 955.2	16.92
	H	26 191	**26 030**	26 468.8	220.74	18 882	**18 829**	19 468.3	73.13	15 936	**15 848**	16 236.6	20.01
	I	**28 992**	29 154	29 596.6	202.43	20 682	**20 610**	21 120.3	132.23	16 554	**16 477**	16 809.6	32.49
	J	26 607	**26 204**	26 916.2	231.03	**18 958**	19 478	19 804.6	72.65	**15 678**	15 951	16 286.3	20.52
500	A	29 536	**28 742**	29 323.6	400.63	21 702	**21 585**	21 758.2	121.52	17 966	**17 840**	18 064.6	36.50
	B	27 370	**27 335**	27 711.1	332.67	**20 523**	20 762	21 082.2	92.81	**17 161**	17 574	17 898.5	28.87
	C	31 494	**31 108**	31 692.7	440.35	23 034	**22 738**	23 108.7	177.94	18 529	**18 498**	18 758.4	45.04
	D	31 752	**30 794**	31 428.4	426.51	22 774	**22 737**	23 032.2	163.36	**18 307**	18 573	18 838.4	44.01
	E	31 555	**30 674**	31 371.7	398.15	22 775	**22 480**	22 812.1	192.74	18 351	**18 335**	18 608.8	47.78
	F	**28 957**	29 258	29 812.3	263.14	21 745	**21 679**	22 022.8	121.52	18 101	**17 976**	18 263.6	36.48
	G	27 492	**27 198**	27 958.2	306.38	**20 325**	20 617	20 983.3	93.22	17 697	**17 600**	17 894.2	29.68
	H	37 185	**36 857**	37 361.1	600.00	26 250	**25 383**	25 968.7	347.42	19 633	**19 619**	19 881.4	79.68
	I	31 612	**31 045**	31 536.0	316.74	22 472	**22 442**	23 083.6	172.67	18 349	**18 322**	18 618.4	47.21
	J	**31 412**	31 423	31 877.9	425.56	22 756	**22 517**	23 054.5	164.47	18 446	**18 445**	18 796.7	44.57

small instances, with n in $\{20, 30, 40, 50, 60\}$, and *large* instances, with n in $\{100, 200, 300, 400, 500\}$. The vehicle capacity Q takes values in $\{10, 15, 20, 25, 30, 35, 40, 45, 50, 1000\}$. There are 10 instances for each combination of n and Q, named A to J. The data set is available from the web site http://webpages.ull.es/users/hhperez/PDsite.

Due to the large number of instances, and in order not to overload the tables, we only report results for three significant values of Q: 10, 20 and 40. Note that, for a given number of customers n, the smaller Q is, the more difficult the problem gets. On the other hand, for all instances, when $Q=1000$ the solution of the 1-PDTSP coincides with the solution of the TSP. Therefore, the three chosen values for Q are the most restrictive ($Q=10$), a rather relaxed one ($Q=40$), and one intermediate ($Q=20$).

The optimal solution for all small instances is known and was obtained by means of the branch-and-cut algorithm in [6]. For large instances, only the approximate solutions given in [1] are available. We compare our solutions with the optimal ones, in the case of small instances, and with the best known approximate solutions, in the case of large instances.

We ran our hybrid algorithm 25 times over each instance, and report the best and average solution values found over the 25 runs. The results are summarized in Tables 1 and 2. Column headings stand for:

- n: Number of customers.
- Name: Instance name.
- Q: Vehicle capacity.
- OPT: Optimal solution (only for small instances).
- *UB*2: Best approximate solution provided in [1] (only for large instances).
- H-best: Best solution value found by the hybrid algorithm.
- H-aver.: Average solution value given by the hybrid algorithm.
- Time: Average runtime, in seconds, for the hybrid algorithm.

Table 1 shows the results for small instances, i.e., those with up to 60 customers. The best solution found by our hybrid heuristic coincides with the optimum for 145 out of the 150 instances in the table (the best algorithm in [1] does so for 118 out of the 150 instances). These coincidences are highlighted in bold. Most of the few cases where the optimum was not attained correspond to instances with 60 customers and $Q = 10$. Those are also the a priori most difficult instances in this table, as explained before, and the ones that require more computing time (around 2.7 s for the hardest). For the other instances, the computing time is generally under 1 s. Note as well that, in 98 cases, also the average solution given by the hybrid algorithm is equal to the optimum. This means that the algorithm finds the optimum in all the 25 runs executed over those instances.

Table 2 shows the results for large instances, with n ranging from 100 to 500. The aim of the table is to compare our heuristic solutions with those given in [1], since the optimal values are not known. As mentioned before, two heuristic methods, called *UB*1 and *UB*2, are described and tested in [1]. The computational results in that article show that the second heuristic (*UB*2) clearly outperforms the first one (*UB*1) in terms of solution quality. In particular, for all the instances in Table 2, the solution provided by *UB*2 is better than the one given by *UB*1. Therefore, we compare our hybrid method only with *UB*2. For each instance, we highlight in bold the best solution found, either by *UB*2 or by our hybrid algorithm (in column H-best). It can be observed that H-best is less than or equal to *UB*2 in 113 out of the 150 cases, providing the best solution known for those instances until the moment. On the other hand, only in 13 cases the average solution (in column H-aver.) given by the hybrid method is better than *UB*2. This indicates the importance of running several times the hybrid heuristic over each instance, or, alternatively, augmenting the number of internal iterations of the GRASP loop if only one run is performed.

Regarding computing time, observe that it augments when n increases and Q decreases. For a given value of n, the instances with $Q = 10$ require much more computing time than those with $Q = 20$ and 40. The time limit of 600 s is reached in some instances with $n = 500$ and $Q = 10$, but this can still be considered a reasonable time for this type of difficult routing problems. For all other instances in the table, the process stops because of the limit on the number of iterations in the GRASP.

In Tables 1 and 2 we have only reported the running times of the algorithm we propose, and not those of the algorithm *UB*2 presented in [1]. Table 3 intends to show how the two algorithms compare in terms of computational time. Column *UB*2 gives the average time given in [1], over the 10 instances for each value of n and Q, while column H gives the average computing time of the hybrid heuristic on the same instances. Notice that these times are not directly comparable because the ones in column *UB*2 were obtained on a different computer (an AMD PC at 1.3 GHz under Windows 98). We have made some computational tests in order to compare the two computers, and we have concluded that our PC is approximately 2.4 times faster than the one used in [1]. With this figure in mind, we can derive from Table 3 that the hybrid algorithm is quicker than *UB*2 for instances with $Q = 20$ and 40, but it is not so for instances with $Q = 10$. This relation holds in general for all problem sizes, although it is more noticeable for large values of n, where running times get more appreciable. Nevertheless, the slightly larger running times of the hybrid heuristic respect to *UB*2 for some instances are compensated with an improvement in the solution quality. This is due to the diversification mechanism implicit in our heuristic that allows to escape from local minima, which does not have an equivalent in the deterministic scheme described in [1]. In fact, we can expect the hybrid heuristic will find better solutions if it runs for a longer time, while the same is not expected to happen with *UB*2.

Table 3
Time comparison.

n	*Q*	*UB*2	*H*
Small instances			
20	10	0.19	0.07
	20	0.05	0.03
	40	0.05	0.02
30	10	0.61	0.33
	20	0.28	0.08
	40	0.06	0.05
40	10	2.21	0.80
	20	0.45	0.15
	40	0.12	0.09
50	10	6.60	1.51
	20	1.81	0.34
	40	0.43	0.17
60	10	8.18	2.40
	20	3.34	0.49
	40	0.56	0.26
Large instances			
100	10	29.38	8.85
	20	18.74	2.22
	40	3.14	0.69
200	10	151.96	41.76
	20	123.89	17.37
	40	51.30	4.35
300	10	285.56	117.86
	20	364.38	50.90
	40	194.84	12.89
400	10	453.64	220.40
	20	423.05	91.73
	40	290.91	23.92
500	10	708.95	391.47
	20	836.03	164.77
	40	350.61	43.98

Fig. 2 depicts the evolution of the objective function value respect to the number of iterations for an instance with 60 customers and a vehicle with capacity equal to 10. The graphic shows how the objective value is progressively updated each time a better solution is found during the GRASP loop. Then the post-optimization procedure is applied and it manages to further improve the solution, giving the optimum for this particular instance. This can be considered the common behavior of the hybrid heuristic, although it is also possible, due to the random component, to find the best solution already in the first iterations, or that the post-optimization phase fails to improve the solution given by the GRASP loop.

Finally, Table 4 illustrates the effect of the post-optimization phase, consisting of VND_2, on the solution quality for the instances with 100 customers. Column heading n-improv. stands for the number of runs, out of 25, where the post-optimization succeeds to improve the solution given by the GRASP loop, while column heading %-improv. stands for the average percentage of improvement obtained. The results show that the post-optimization phase is effective in most cases, although it produces a limited solution improvement (typically under 1%). Moreover, it seems to be more useful on those instances with the most restrictive vehicle capacity ($Q = 10$).

Considering only the percentage of improvement over the GRASP solution, it might appear that the post-optimization phase could be removed. However, in terms of absolute value, the obtained improvement makes the difference between beating the *UB*2 approach or not. In this sense, for 22 out of the 30 instances in Table 4 the cost of the solution generated by the GRASP loop is less than or equal to the cost of the solution obtained from the *UB*2 approach. This figure raises to 26 out of 30 instances when the post-optimization phase is applied.

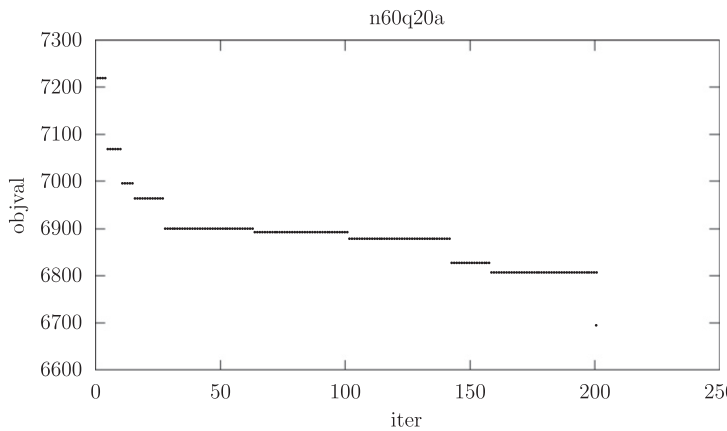

Fig. 2. Evolution of the objective value versus number of iterations for an instance with $n=60$ and $Q=20$.

Table 4
Post-optimization contribution to the final solution.

n	Name	$Q=10$		$Q=20$		$Q=40$	
		n-improv.	%-improv.	n-improv.	%-improv.	n-improv.	%-improv.
100	A	23	0.45	15	0.30	7	0.11
	B	23	0.42	24	0.36	16	0.20
	C	24	0.33	23	0.38	18	0.28
	D	24	0.35	24	0.57	17	0.34
	E	25	0.22	18	0.41	16	0.33
	F	23	0.20	16	0.37	23	0.20
	G	23	0.47	22	0.33	12	0.17
	H	21	0.22	24	0.28	18	0.31
	I	17	0.24	21	0.32	13	0.17
	J	25	0.43	22	0.29	13	0.15

4. Conclusions

This article presents an efficient hybrid heuristic algorithm to tackle the one-commodity pickup-and-delivery traveling salesman problem. Hybrid metaheuristics have received considerable attention in recent years, and have proven to be highly useful for solving difficult optimization problems. The method we propose uses the basic structure of GRASP to better sampling the solution space. The GRASP local search phase is replaced by a VND with two neighborhood structures based on the classical 2-opt and 3-opt operators, conveniently modified to cope with the feasibility requirement for 1-PDTSP tours. A second VND procedure is applied to the best solution found by the GRASP in order to further improve it. The neighborhood structures in this second VND are related to vertex-exchange movements. Both VND algorithms work as intensification mechanisms, while the random multi-start provides diversification to the search.

The performance of the GRASP/VND hybrid heuristic is evaluated on benchmark instances with different sizes and degrees of difficulty. The results are very satisfactory, specially when compared to the ones in the literature. Moreover, the algorithm appears robust in terms of quality and computational effort. The conclusion is that the proposed method is competitive with the other approaches previously presented, managing to find the optimal solution in 96.7% of the small instances tested, and providing the best solution until now in 75.3% of the large instances.

These results are very encouraging and, as future research work, we consider the application of a similar hybrid methodology to related pickup-and-delivery problems. In particular, a natural extension of the 1-PDTSP is the *multi-commodity pickup-and-delivery traveling salesman problem*. In this problem, there are m different products and each customer may supply and/or demand an amount of each product. As in the 1-PDTSP, a capacitated vehicle must visit each customer exactly once, and the objective is to find a minimum length tour. Another interesting variant is the *one-to-one multi-commodity pickup-and-delivery traveling salesman problem*, where each product has exactly one origin and one destination customer. Based on our experience with the 1-PDTSP, we think that the design of hybrid heuristics for these problems is a worth pursuing research direction.

Acknowledgment

This work has been partially supported by "Ministerio de Educación y Ciencia'', Spain (research project MTM2006-14961-C05-03).

References

[1] Hernández-Pérez H, Salazar-González JJ. Heuristics for the one-commodity pickup-and-delivery traveling salesman problem. Transportation Science 2004;38:245–55.
[2] Anily S, Bramel J. Approximation algorithms for the capacitated traveling salesman problem with pickups and deliveries. Naval Research Logistics 1999;46:654–70.
[3] Chalasani P, Motwani R. Approximating capacitated routing and delivery problems. SIAM Journal on Computing 1999;28:2133–49.
[4] Wang F, Lim A, Xu Z. The one-commodity pickup and delivery travelling salesman problem on a path or a tree. Networks 2006;48:24–35.
[5] Hernández-Pérez H. Traveling salesman problems with pickups and deliveries. Dissertation, University of La Laguna, Spain; 2004.
[6] Hernández-Pérez H, Salazar-González JJ. A branch-and-cut algorithm for a traveling salesman problem with pickup and delivery. Discrete Applied Mathematics 2004;145:126–39.
[7] Savelsbergh MWP, Sol M. The general pickup and delivery problem. Transportation Science 1995;29:17–29.
[8] Parragh SN, Doerner K, Hartl RF. A survey on pickup and delivery models Part I: transportation between customers and depot. Working paper, Chair of Production and Operations Management, University of Vienna; 2006.
[9] Parragh SN, Doerner K, Hartl RF. A survey on pickup and delivery models Part II: transportation between pickup and delivery locations. Working paper, Chair of Production and Operations Management, University of Vienna; 2006.
[10] Berbeglia G, Cordeau J-F, Gribkovskaia I, Laporte G. Static pickup and delivery problems: a classification scheme and survey. TOP 2007;15:45–7.
[11] Feo TA, Resende MGC. Greedy randomized adaptative search procedures. Journal of Global Optimization 1995;6:109–33.
[12] Festa P, Resende MGC. GRASP: an annotated bibliography. In: Ribeiro CC, Hansen P, editors. Essays and surveys in metaheuristics. Dordrecht: Kluwer Academic Publishers; 2002.
[13] Resende MGC, Ribeiro CC. Greedy randomized adaptative search procedures. In: Glover F, Kochenberger GA, editors. Handbook of metaheuristics. Kluwer's international series in operations research and management science. Norwell: Kluwer Academic Publishers; 2002.
[14] Mladenović N, Hansen P. Variable neighborhood search. Computers & Operations Research 1997;24:1097–100.
[15] Lin S. Computer solutions of the traveling salesman problem. Bell System Technical Journal 1965;44:2245–69.
[16] Johnson DS, McGeoch LA. The traveling salesman problem: a case study in local optimization. In: Aarts EJL, Lenstra JK, editors. Local search in combinatorial optimization. Chichester, UK: Wiley; 1997.
[17] Lin S, Kernighan BW. An effective heuristic algorithm for the traveling salesman problem. Operations Research 1973;21:498–516.
[18] Mosheiov G. The travelling salesman problem with pick-up and delivery. European Journal of Operational Research 1994;79:299–310.