Heuristics for the One-Commodity Pickup-and-Delivery Traveling Salesman Problem

Article *in* Transportation Science · May 2004

DOI: 10.1287/trsc.1030.0086 · Source: DBLP

2 authors:

Hipólito Hernández-Pérez
Universidad de La Laguna
15 PUBLICATIONS **542** CITATIONS

SEE PROFILE

Juan José Salazar González
Universidad de La Laguna
164 PUBLICATIONS **3,697** CITATIONS

SEE PROFILE

TRANSPORTATION SCIENCE
Vol. 38, No. 2, May 2004, pp. 245–255
ISSN 0041-1655 | EISSN 1526-5447 | 04 | 3802 | 0245

DOI 10.1287/trsc.1030.0086

Heuristics for the One-Commodity Pickup-and-Delivery Traveling Salesman Problem

Hipólito Hernández-Pérez, Juan-José Salazar-González
DEIOC, Facultad de Matemáticas, Universidad de La Laguna, 38271 La Laguna, Tenerife, Spain
{hhperez@ull.es, jjsalaza@ull.es}

This paper deals with a generalisation of the well-known *traveling salesman problem* (TSP) in which cities correspond to customers providing or requiring known amounts of a product, and the vehicle has a given upper limit capacity. Each customer must be visited exactly once by the vehicle serving the demands while minimising the total travel distance. It is assumed that any unit of product collected from a pickup customer can be delivered to any delivery customer. This problem is called *one-commodity pickup-and-delivery TSP* (1-PDTSP). We propose two heuristic approaches for the problem: (1) is based on a greedy algorithm and improved with a k-optimality criterion and (2) is based on a branch-and-cut procedure for finding an optimal local solution. The proposal can easily be used to solve the classical "TSP with pickup-and-delivery," a version studied in literature and involving two commodities. The approaches have been applied to solve hard instances with up to 500 customers.

Key words: pickup and delivery; traveling salesman; heuristics; branch and cut
History: Received: August 2002; revision received: March 2003; accepted: March 2003.

1. Introduction

This paper presents two heuristic algorithms for the 1-PDTSP. The problem is closely related to the well-known TSP, thus, a finite set of cities is given, and the travel distance between each pair of cities is assumed to be known. In addition, one specific city is considered to be a *depot* of a vehicle, while the other cities are identified as *customers* and are divided into two groups according to the type of service required (delivery or pickup). Each *delivery customer* requires a given amount of the product and each *pickup customer* provides a given amount of the product. Any amount of the product collected from a pickup customer can be supplied to a delivery customer. It is assumed that the vehicle has a fixed upper limit *capacity* and must start and end the route at the depot. The 1-PDTSP then calls for a minimum distance route for the vehicle to satisfy the customer requirements without ever exceeding the vehicle capacity. The route must be a Hamiltonian tour through all the cities. As in most of the papers on similar routing problems, we will assume that the travel distances are symmetric, even if most of the ideas presented here could be extended to also work on asymmetrical instances.

We now introduce the notation useful in this paper. The depot will be denoted by 1 and each customer by i $(i = 2, \ldots, n)$. $V := \{1, 2, \ldots, n\}$ 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 going between i and j is given. It is also given a nonzero demand q_i associated with each customer i, where $q_i < 0$ if i is a delivery customer and $q_i > 0$ if i is a pickup customer. Let $K := \max\{\sum_{i \in V: q_i > 0} q_i, -\sum_{i \in V: q_i < 0} q_i\}$. The capacity of the vehicle is represented by Q and is assumed to be a positive number. Note that typically $Q \leq K$ on a 1-PDTSP instance. Clearly, 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 remaining amount of product to ensure product conservation. Even more, in this basic version, we will assume that the depot can also provide the vehicle with the necessary load for guaranteeing the feasibility of some routes that need a nonempty (or nonfull) load of the vehicle when coming out from the depot. Indeed, notice that a 1-PDTSP instance could admit a solution for the vehicle even if there is no route with the vehicle leaving the depot with an empty (or full) load. Still, it is possible to solve the 1-PDTSP with this additional requirement by using an algorithm for another basic 1-PDTSP with the depot replaced by two dummy customers. Another interesting observation from the definition of the problem is that when $T \subset E$ is a set of edges that can be oriented defining a feasible circuit $\overrightarrow{T}$ for a 1-PDTSP instance, then the circuit $\overleftarrow{T}$ obtained by orienting the edges in the other sense is also a feasible route for the vehicle of the same instance (see Hernández-Pérez and Salazar-González 2004 for details).

The 1-PDTSP was introduced in Hernández-Pérez and Salazar-González (2004) together with an exact

approach applied to solve instances with up to 40 customers, but there are in literature many other problems closely related. We next point out some of them, starting with the problems involving one commodity and then the related problems involving several commodities.

Chalasani and Motwani (1999) study the special case of the 1-PDTSP, where the delivery and pickup quantities are all equal to one unit. This problem is called *Q-delivery TSP* if Q is the capacity of the vehicle. Anily and Bramel (1999) consider the same problem with the name *capacitated traveling salesman problem with pickups and deliveries* (CTSPPD). They propose heuristic algorithms, which could be applied to a 1-PDTSP instance with general demands if the Hamiltonian requirement on the route were relaxed, or directly to a 1-PDTSP instance if the demand values are $+1$ or -1. In other words, as noticed in Anily and Bramel (1999), CTSPPD can be considered the "splitting version" of the 1-PDTSP. Moreover, Anily and Bramel (1999) give an important application of the CTSPPD in the context of inventory repositioning, and this application leads to the 1-PDTSP if the vehicle is required to visit each retailer exactly once, an important constraint to guarantee a satisfactory service policy when the customer is external to the firm.

Anily and Hassin (1992) introduce the *swapping problem*, a more general problem, where several commodities must be transported from many origins to many destinations with a vehicle of limited capacity following a non-necessary Hamiltonian route, and where a commodity can be temporarily stored in an intermediate customer. This last feature is named *preemption* or *drop* in the process of transportation. The swapping problem on a line is analyzed in Anily et al. (1999). In the *capacitated dial-a-ride problem* (CDARP) (also named *single-vehicle many-to-many dial-a-ride problem*), there is a one-to-one correspondence between pickup customers and delivery customers, and the vehicle should move one unit of a commodity (say, a person) from its origin to its destination with a limited capacity Q (see, e.g., Psaraftis 1983, Guan 1998). When $Q = 1$, the nonpreemptive CDARP is known as the *stacker crane problem* (see, e.g., Frederickson et al. 1978). When there is no vehicle capacity, the nonpreemptive CDARP is called the *pickup and delivery traveling salesman problem* (PDTSP) (see, e.g., Kalantari et al. 1985, Kubo and Kasugai 1990, Healy and Moll 1995, Renaud et al. 2000, Renaud et al. 2002). In Kalantari et al. (1985), the PDTSP is referred to as *traveling salesman problem with pickup and delivery* (TSPPD) and in Healy and Moll (1995), the PDTSP is referred to as CDARP. The PDTSP is a particular case of the *TSP with precedence constraints* (see, e.g., Bianco et al. 1994). Another related problem is the *TSP with backhauls* (TSPB) (see, e.g., Gendreau et al. 1997), where an uncapacitated vehicle must visit all the delivery customers before visiting a pickup customer. See Savelsbergh and Sol (1995) for other references and variants (including time windows, and so on) of the so-called *general pickup and delivery problem*.

Another known problem is the TSPPD, introduced by Mosheiov (1994), and where the product collected from pickup customers must be transported to the depot and the product delivery to the demand customers should be transported from the depot, all done using a capacitated vehicle on a Hamiltonian route. Seen from this point of view, the TSPPD concerns a many-to-one commodity and another one-to-many different commodity, and a classical application is the collection of empty bottles from the customers being delivered to a warehouse and full bottles being delivered from the warehouse to the customers. An important difference when comparing TSPPD with 1-PDTSP is that PDTSP is feasible if, and only if, the vehicle capacity is at least the maximum of the total sum of the pickup demands and of the total sum of the delivery demands (i.e., $K \leq Q$). This constraint is not required for the feasibility of the 1-PDTSP in which the vehicle capacity Q could even be equal to the biggest customer demand $\max_{i=1}^{n} |q_i|$. As mentioned by Mosheiov (1994), a TSPPD instance can be assumed to be in "standard form," which means that the capacity of the vehicle coincides with the total sum of products collected in the pickup customers and with the total sum of product delivered in the delivery customers (i.e., $K = Q$). Hence, it is always possible to assume that in a feasible TSPPD route, the vehicle starts at the depot fully loaded, it delivers and picks up amounts from the customers, and finishes at the depot again fully loaded. This condition of traversing the depot with a full (or empty) load is also presented in the other problems, like the CTSPPD, where the vehicle cannot go immediately from the depot to a pickup customer, but it does not necessarily occur in general with the 1-PDTSP as observed.

Anily and Mosheiov (1994) present approximation algorithms for the TSPPD, here renamed *TSP with delivery and backhauls* (TSPDB), and Gendreau et al. (1999) propose several heuristics tested on instances with up to 200 customers. Baldacci et al. (1999) deal with the same problem, here named *TSP with delivery and collection* (TSPDC) *constraints*, and present an exact algorithm based on a two-commodity network flow formulation, which was able to prove optimality of some TSPPD instances with 150 customers, even if it was not able to prove optimality of other TSPPD instances with 25 customers. Hernández-Pérez and Salazar-González (2004) present a branch-and-cut algorithm for the exact solution of TSPPD instances

Table 1 Characteristics of Closely Related Problems

Problem Name	#	Origins-Destinations	Hamiltonian	Pre-emption	Q	Load	References
Swapping Problem	m	many to many	no	yes	1	yes	Anily et al. (1999), Anily and Hassin (1992)
Stacker Crane Problem	m	one to one	no	no	1	yes	Frederickson et al. (1978)
CDARP	m	one to one	no	no	k	yes	Guan (1998), Psaraftis (1983), Savelsbergh and Sol (1995)
PDTSP	m	one to one	yes	no	∞	yes	Kalantari et al. (1985), Kubo and Kasugai (1990), Renaud et al. (2002), Renaud et al. (2000)
TSPPD, TSPDB, TSPDC	2	one to many	yes	no	k	yes	Anily and Mosheiov (1994), Baldacci et al. (1999), Gendreau et al. (1999), Mosheiov (1994)
TSPB	2	one to many	yes	no	∞	yes	Gendreau et al. (1997)
CTSPPD, Q-delivery TSP	1	many to many	no	no	k	yes	Anily and Bramel (1999), Chalasani and Motwani (1999)
1-PDTSP	1	many to many	yes	no	k	no	Hernández-Pérez and Salazar-González (2004)

with up to 75 customers and of 1-PDTSP instances with up to 40 customers.

On the one hand, Chalasani and Motwani (1999) observe that when the vehicle capacity is large enough, then their problem (i.e., the Q-delivery TSP or CTSPPD with $Q = \infty$) is a special case of the TSPPD. This implies that the uncapacitated version of the 1-PDTSP, with the additional requirement that the vehicle must leave the depot with a full load, can be approximated by the methods developed in Mosheiov (1994) and in Anily and Mosheiov (1994). On the other hand, Anily and Bramel (1999) observe that a TSPPD instance can also be transformed into a CTSPPD instance and, therefore, algorithms for solving the 1-PDTSP can be easily adapted to solve the TSPPD. This was the main motivation for our work.

This extensive and nonexhaustive list of single-vehicle problems is summarised in Table 1 with respect to the following features:

Problem name: it is the short name of the problem.

#: number of commodities.

Origins-destinations: it is the number of pickup customers and delivery customers of a commodity ("many to many" if there are several origins and several destinations. "One to one" if there is one origin and one destination, and so on).

Hamiltonian: "yes" means that the problem looks for a Hamiltonian tour, and "no" means that a customer can be visited more than once.

Pre-emption: "yes" means that an intermediate drop is allowed, and "no" means that a unit of product should be loaded once in the vehicle.

Q: it is the assumption on the vehicle capacity (a unit value, a general value k or the uncapacitated situation).

Load: "yes" means that the vehicle should leave the depot with a full or empty load, and "no" means that the initial load should be also computed.

References: some papers on the problem.

An important remark is that some capacitated problems with no Hamiltonian requirement are addressed in some papers by splitting each customer with demand q into q customers with unit demand, and then solving a larger problem with the Hamiltonian requirement. This is, for example, the case of the CTSPPD in Anily and Bramel (1999) and Chalasani and Motwani (1999), where the proposed algorithms are developed for a particular case of the 1-PDTSP with unit demands. Hence, the feature on Hamiltonianity referred to in Table 1 regards the original problem with general demand values and not the larger problem with unit demands when $Q < \infty$.

Clearly, the 1-PDTSP is an $\mathcal{NP}$-hard optimization problem in the strong sense because it coincides with TSP when the vehicle capacity is large enough. Additionally, checking if there is a feasible solution of a 1-PDTSP instance is a strongly $\mathcal{NP}$-complete problem. The idea is based on the fact that this decision problem includes the *3-Partitioning Problem* (3PP), which is $\mathcal{NP}$-complete in the strong sense (see Garey and Johnson 1979) and defined as follows. Let us consider a number P and a set I of $3m$ positive integer numbers $p_1, \ldots, p_{3m}$ such that $P/4 < p_i < P/2$ and $\sum_{i \in I} p_i = mP$. The 3PP requires the existence of a partition of I into m disjoint sets $I_1, \ldots, I_m$ such that $\sum_{i \in I_k} p_i = P$ for all $k \in \{1, \ldots, m\}$. For each 3PP instance, there is an associated 1-PDTSP defined by $3m$ pickup customers, each one with a demand p_i, by m delivery customers, each one with demand $-P$, and by a vehicle of capacity P. The problems of checking feasibility for both instances coincide.

Finding good feasible 1-PDTSP solutions is much more complicated than finding good heuristic solutions for the TSP, thus, justifying the aim of this article. Section 2 describes a simple initial heuristic algorithm by extending some TSP procedures. Section 3 describes a more elaborate procedure based on a branch-and-cut algorithm as proposed by Fischetti and Lodi (2002) for general mixed integer programming models. An extensive computational analysis is presented in §4, where 1-PDTSP and TSPPD instances with $n \leq 500$ are heuristically solved.

2. First Heuristic for 1-PDTSP

As mentioned in the introduction, contrary to what happens in the TSP, finding a feasible solution (optimal or not) for the 1-PDTSP is a problem with a hard

computational complexity. Nevertheless, checking if a given TSP solution is feasible for 1-PDTSP is a trivial task as it can be done in linear time on n. Indeed, let us consider a path $\overrightarrow{P}$ defined by the vertex sequence $v_1, \ldots, v_k$ for $k \le n$. Let $l_0(\overrightarrow{P}) := 0$ and $l_i(\overrightarrow{P}) := l_{i-1}(\overrightarrow{P}) + q_{v_i}$ be the load of the vehicle coming out from v_i if it goes inside v_1 with load $l_0(\overrightarrow{P})$. Notice that $l_i(\overrightarrow{P})$ could be a negative. We denote

$$infeas(\overrightarrow{P}) := \max_{i=0}^{k}\{l_i(\overrightarrow{P})\} - \min_{i=0}^{k}\{l_i(\overrightarrow{P})\} - Q$$

the *infeasibility* of the path $\overrightarrow{P}$. This value is independent of the orientation of the path (i.e., $infeas(\overrightarrow{P}) = infeas(\overleftarrow{P})$, where $\overleftarrow{P}$ denotes the path $\overrightarrow{P}$ in the opposite direction), and a path $\overrightarrow{P}$ is said to be *feasible* if $infeas(\overrightarrow{P}) \le 0$. Then, the interval of potential loads of the vehicle going out from v_k is defined by the extreme values

$$\underline{l_k(\overrightarrow{P})} := l_k(\overrightarrow{P}) - \min_{i=0}^{k}\{l_i(\overrightarrow{P})\}$$

$$\overline{l_k(\overrightarrow{P})} := l_k(\overrightarrow{P}) + Q - \max_{i=0}^{k}\{l_i(\overrightarrow{P})\}.$$

Immediately, we also have the interval of potential loads of the vehicle entering v_1 to route $\overrightarrow{P}$

$$\left[\underline{l_k(\overrightarrow{P})} - l_k(\overrightarrow{P}),\ \overline{l_k(\overrightarrow{P})} - l_k(\overrightarrow{P})\right],$$

and of going out from v_1 if the path is routed in the reverse order, defined by

$$\underline{l_1(\overleftarrow{P})} := Q - \overline{l_k(\overrightarrow{P})} + l_k(\overrightarrow{P})$$

$$\overline{l_1(\overleftarrow{P})} := Q - \underline{l_k(\overrightarrow{P})} + l_k(\overrightarrow{P}).$$

For example, if $\overrightarrow{P}$ is a path defined by customers v_1, v_2, v_3 with demands $+2, -5, +1$, respectively, and $Q = 10$, then the vehicle can enter and follow $\overrightarrow{P}$ whenever its load is in $[3, 8]$. Immediately, after coming out from the path, the load of the vehicle is in $[1, 6]$ and, therefore, the vehicle can enter and follow the path in the reverse order when its load is in the range [4,9].

Given two disjoint feasible paths $\overrightarrow{P}$ and $\overrightarrow{R}$, the first from v_{i_1} to v_{i_k} and the second from v_{j_1} to v_{j_s}, it is simple to check if they can be merged in a single feasible path by considering $\overrightarrow{P}$, $\overrightarrow{R}$ and the arc (v_{i_k}, v_{j_1}). Indeed, the merged path will be feasible if

$$\left[\underline{l_{i_k}(\overrightarrow{P})},\ \overline{l_{i_k}(\overrightarrow{P})}\right] \cap \left[\underline{l_{j_s}(\overrightarrow{R})} - l_{j_s}(\overrightarrow{R}),\ \overline{l_{j_s}(\overrightarrow{R})} - l_{j_s}(\overrightarrow{R})\right] \neq \varnothing.$$

Moreover, the intersection interval gives the range of potential loads for the vehicle on the merged path, hence, it is not necessary to recompute from scratch this range for the new path. See Figure 1 for an illustration. The load ranges of the vehicle going in and out of each path are represented as vertical segments, emphasising the interval of the merged path in bold lines.

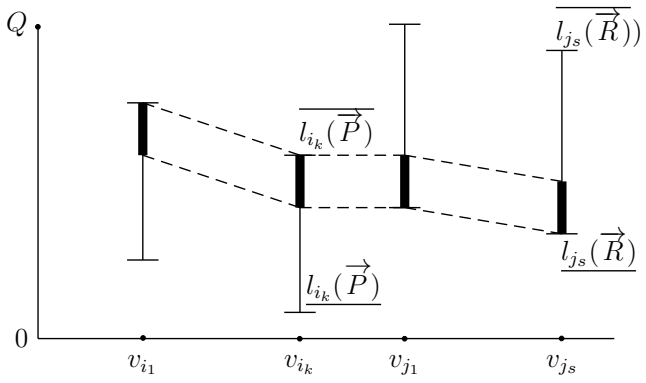

Figure 1 Merging Two Feasible Paths

We will denote the routing cost of the path $\overrightarrow{P}$ by $cost(\overrightarrow{P})$.

This idea allows us to extend different TSP greedy procedures to (possibly) build an initial 1-PDTSP solution, and our initial procedure is an adaptation of the classical TSP *nearest neighbor* algorithm. However, because this simple algorithm hardly ever finds a feasible 1-PDTSP solution when the vehicle capacity is tight, we first redefine the travel costs for this initial routine. The aim is to encourage the use of edges connecting customers of different types by slightly penalising the using of edges $[i, j]$ with $q_i = q_j$ as follows:

$$d_{ij} := \begin{cases} c_{ij} + C(2Q - |q_i - q_j|) & \text{if } |q_i + q_j| \le Q, \\ \infty & \text{otherwise,} \end{cases}$$

where $C := (K - Q)\sum_{[i,j]\in E} c_{ij}/(10nQ)$ is a constant that depends on the scales of the costs, demands, and vehicle capacity. Then, starting from a given initial vertex v_1, the algorithm iteratively extends a partial path. At each iteration, the partial path from v_1 to v_k is extended by choosing a new vertex v_{k+1} among the closest to v_k according to d_{ij} and such that the new partial path through $v_1, \ldots, v_k, v_{k+1}$ is feasible. If there is not a vertex v_{k+1} leading to a feasible extended path, then v_{k+1} is selected minimising the infeasibility of the extended path. This procedure, named *greedy algorithm* in the sequel, could finish with a tour $\overrightarrow{T}$, which can be unfeasible. The aim of considering $|q_i - q_j|$ instead of $|q_i + q_j|$ is to encourage a route going from a customer i to a customer j with demand q_j as different from q_i as possible. Other similar formulas based on c_{ij}, $|q_i + q_j|$ and $|q_i - q_i|$ to define d_{ij} were empirically tested and the above referred provided the best results on our benchmark instances.

Tour improvement procedures are adaptations of the *2-opt* and *3-opt exchanges of edges* described in Lin (1965) and Johnson and McGeoch (1997) for the TSP. These implementations avoid looking at all possible exchanges for one that shortens the tour, in the spirit of work of Lin and Kernighan (1973), and are based on a data structure that quickly identifies allowable candidates for the exchanges (see also Helsgaun 2000) for a sophisticated variant. In our implementation, for each vertex i, the data structure stores a list of 15 remaining vertices j in order of increasing distance d_{ij}, thus, defining allowable edges $[i, j]$ to be inserted in exchanges.

The execution of the proposed approach is repeated for each initial vertex v_1 when $n < 100$. For larger instances, to keep the required computational effort small, only 100 vertices i with biggest $|q_i|$ are considered as initial vertices v_1. Therefore, this heuristic can be seen as a multistart approach. In our experiments, we have observed that the loss of quality involved in using truncated neighbor lists is not a major issue in practice, as also noticed in Johnson and McGeoch (1997). A pseudocode describing the overall procedure is next given.

Step 1. Let N_0 be the set of initial vertices in the multistart procedure.

Step 2. Select $v_1 \in N_0$, apply the *greedy algorithm* starting from v_1 and let $\overrightarrow{T}$ be the output.

Step 3. If $\overrightarrow{T}$ is feasible 1-PDTSP solution, set $\overrightarrow{T}^* := \overrightarrow{T}$ and go to Step 4; otherwise, set $\overrightarrow{T}' := T$ and go to Step 5.

Step 4. Apply *2-opt procedure* and *3-opt procedure* to $\overrightarrow{T}^*$, obtaining a new tour $\overrightarrow{T}$ minimising $infeas(\overrightarrow{T})$, subject to $infeas(\overrightarrow{T}) < 2K/n$ and $cost(\overrightarrow{T}) \le cost(\overrightarrow{T}^*)$. If no $\overrightarrow{T}$ is found, go to Step 6; otherwise, go to Step 3.

Step 5. Apply *2-opt procedure* and *3-opt procedure* over $\overrightarrow{T}'$, obtaining a new tour $\overrightarrow{T}$ minimising $infeas(\overrightarrow{T})$, subject to $infeas(\overrightarrow{T}) < infeas(\overrightarrow{T}')$ and $cost(\overrightarrow{T}) \le cost(\overrightarrow{T}^*)$. If no $\overrightarrow{T}$ is found, go to Step 6; otherwise, go to Step 3.

Step 6. If $\overrightarrow{T}^*$ improves the current best tour $\overrightarrow{T}^1$, update $\overrightarrow{T}^1$. Set $N_0 := N_0 \backslash \{v_1\}$. If $N_0 \neq \varnothing$, go to Step 1; otherwise, stop with the output $\overrightarrow{T}^1$.

3. Incomplete Optimisation Algorithm for 1-PDTSP

The branch-and-cut algorithm proposed in Hernández-Pérez and Salazar-González (2004) was able to solve to optimality instances with up to 40 vertices. When trying to solve bigger instances, the same approach is not expected to finish within a reasonable computing time. Nevertheless, as proposed in Fischetti and Lodi (2002) for solving general mixed integer linear programming (ILP) models, one could use an available exact procedure to find the best solution in a restricted feasible region within an iterative framework.

Given two integer numbers h and k, let us assume that a 1-PDTSP tour T^h is known, and let $[x_e^h: e \in E]$ be its incident vector and E^h the edges in T^h. We define *k-neighborhood* of T^h the set $\mathcal{N}(T^h, k)$ of all 1-PDTSP solutions that differ from T^h in no more than k edges. In other words, $\mathcal{N}(T^h, k)$ is the set of the solutions of the ILP model plus the additional constraint

$$\sum_{e \in E^h} (1 - x_e) \le k. \tag{1}$$

Note that $\mathcal{N}(T^h, 0) = \mathcal{N}(T^h, 1) = \{T^h\}$, and $\mathcal{R} := \mathcal{N}(T^h, n)$ is the set of all feasible 1-PDTSP tours. One is then interested in developing a procedure to find the best solution in $\mathcal{N}(T^h, k)$ for some k. The first heuristic described in §2 contains an improvement procedure that iteratively finds a good solution T^{h+1} exploring (partially) $\mathcal{N}(T^h, k)$ for $k = 3$. The initial tour T^1 is provided by the heuristic procedure described in §2. Clearly, the direct extension of this improvement procedure to deal with some $k > 3$ could find better solutions, but it could require too much computational effort. Still, by modifying the branch-and-cut code in Hernández-Pérez and Salazar-González (2004) to take into account constraint (1), one could expect to succeed exploring $\mathcal{N}(T^h, k)$ for some $3 < k < n$.

Let us denote T^h, the tour obtained with the h-th call to the branch-and-cut algorithm. As noted by Fischetti and Lodi (2002), the neighborhoods $\mathcal{N}(T^h, k)$ for all T^h are not necessarily disjoint subsets of $\mathcal{R}$. Nevertheless, it is quite straightforward to also modify the branch and cut to avoid searching in already explored regions. To this end, the branch-and-cut algorithm at iteration h can be modified to take into account the original ILP model, constraint (1), and the additional inequalities

$$\sum_{e \in E^l} (1 - x_e) \ge k + 1 \tag{2}$$

for all $l = 1, \ldots, h-1$. Then, at iteration h, it explores a totally different region, defined by $\mathcal{N}(T^h, k) \backslash \bigcup_{l=1}^{h-1} \mathcal{N}(T^l, k)$. The last iteration is the one where no new solution was found.

Fischetti and Lodi (2002) also noticed that by exploring the remaining feasible 1-PDTSP region (i.e., $\mathcal{R} \backslash \bigcup_{l=1}^{h} \mathcal{N}(T^l, k)$ if iteration h is the last iteration), the procedure would be an exact method. Because the remaining region tends to create a difficult problem for a branch-and-cut algorithm on medium-large instances and because we are only interested in good solutions in short time, our approach does not execute this additional exploration.

Solving each iteration could be, in some cases, a difficult task in practice, hence, Fischetti and Lodi

(2002) suggest the imposition of a time limit on each branch-and-cut execution and the modification of the value of k when it is achieved. In our implementation, we found better results by reducing the number of variables given to the branch-and-cut solver. More precisely, instead of starting each branch-and-cut execution with all possible variables, we only allow the fractional solutions to use variables associated with the subset E' of promising edges. Subset E' is initialised with the 10 shortest edges $[i, j]$, according to the distance d_{ij} described in §2, for each vertex i. In our experiments, we did not find advantages in enlarging E' even with the new edges in E^h of the current solution. In this way, the branch-and-cut execution of iteration h will not explore $\mathcal{N}(T^h, k)$.

Also, to avoid heavy branch-and-cut executions, we impose a limit of, at most, six node-deep levels when going down in the decision tree exploration. This idea uses the fact that the applied branch-and-cut algorithm, as mentioned in Hernández-Pérez and Salazar-González (2004), tends to explore a large number of nodes, each one in a short time, hence, the idea tries to diversify the fractional solutions feeding the primal heuristic described in the next paragraph. Whenever the branch-and-cut execution at iteration h ends with a tour T^{h+1} better than T^h, the next iteration is performed.

An element of primary importance for this more elaborated heuristic is the *primal heuristic* applied inside the branch-and-cut algorithm. It is a procedure to find a feasible 1-PDTSP solution by using the information in the last fractional solution, and it is executed every six iterations of the cutting plane approach and before every branching phase of the branch-and-cut procedure. To implement it, we have slightly modified the heuristic described in §2 to consider a given fractional solution $[x_e^*: e \in E']$, where E' is the set of edges with a variable in the model. More precisely, the new procedure to build an initial 1-PDTSP solution chooses edges from $E^* := \{e \in E': x_e^* > 0\}$ to be inserted in a partial tour T. The partial tour T is initialised with n degenerated paths of G, each one P_i consisting in only one vertex v_i. The edges are chosen from E^* in decreasing order of x_e^* whenever they merge two paths into a feasible 1-PDTSP one. If E^* is empty before T is a Hamiltonian tour, then the initial greedy procedure of §2 continues. The final tour is improved by running the 2-opt and 3-opt procedures. Observe that the generated solution could also contain edges from $E\backslash E'$, and even so, it could not satisfy constraint (1). Therefore, the tour T^{h+1} obtained at iteration h (not the last) from the tour T^h is not necessarily in $\mathcal{N}(T^h, k)$, and we have even observed in our experiments that, in some cases, $\mathcal{N}(T^h, k) \cap \mathcal{N}(T^{h+1}, k) = \varnothing$. This last observation suggests that considering constraint (2) in our branch-and-cut executions is not so relevant for our heuristic aims.

To further speed the execution of the branch-and-cut solver, one could replace (1) by $4 \leq \sum_{e \in E^h}(1 - x_e) \leq k$, since the primal heuristic procedure almost guarantees that T^h is the best tour in $\mathcal{N}(T^h, 3)$. When k is a small number (as in our implementation, where $k = 5$), we have also experimented with the replacement of each branch-and-cut execution with the previous constraint by a sequence of branch-and-cut executions, each one with the equality $\sum_{e \in E^h}(1 - x_e) = l$, for each $l = 4, 5, \ldots, k$, going to the next iteration as soon as a better tour T^{h+1} is found. We did not find in our experiments any clear advantage in implementing these ideas, hence, each iteration consists of one branch-and-cut execution solving ILP model with the additional constraint (2).

4. Computational Results

The heuristic algorithms described in §§2 and 3 have been implemented in ANSI C, and ran on a personal computer AMD 1,333 MHz. under Windows 98. The software CPLEX 7.0 was the linear programming (LP) solver in our branch-and-cut algorithm.

We tested the performance of the approaches both on the 1-PDTSP and on the TSPPD using the random generator proposed in Mosheiov (1994) as follows. We have generated $n - 1$ random pairs in the square $[-500, 500] \times [-500, 500]$, each one corresponding to the location of a customer and associated with a random demand in $[-10, 10]$. The depot was located in (0,0) with demand $q_1 := -\sum_{i=2}^{n} q_i$. The travel cost c_{ij} was computed as the Euclidean distance between i and j. We generated 10 random instances for each value of n in $\{20, 30, \ldots, 60, 100, 200, \ldots, 500\}$ and for each different value of Q in $\{10, 15, \ldots, 45, 1{,}000\}$. The solution of the 1-PDTSP instances with $Q = 1{,}000$ coincided with the optimal TSP tour in all the generated instances. The case $Q = 10$ was the smallest capacity because for each n, there is a customer in an instance with demand ± 10. Also, for each value of n, 10 TSPPD instances were generated, each one with the smallest possible capacity

$$Q = \max\left\{\sum_{q_i > 0:\, i \in V\backslash\{1\}} q_i, -\sum_{q_i < 0:\, i \in V\backslash\{1\}} q_i\right\}.$$

Table 2 shows the results for the small instances ($n \leq 60$) and Table 3 shows the results of the medium-large instances ($100 \leq n \leq 500$). To evaluate the quality of the generated feasible solutions, we have used the exact algorithm described in Hernández-Pérez and Salazar-González (2004) to compute the optimal value LB of the LP relaxation of the integer programming model for all instances, the optimal

Table 2 Statistics on Small Instances of the 1-PDTSP and TSPPD

n	*Q*	*UB1/OPT*	*UB1/LB*	*UB1-t*	*UB2/OPT*	*UB2/LB*	*UB2-t*	*B&C*
20	PDTSP	100.00	100.83	0.02	100.00	100.83	0.08	1.0
20	10	100.16	101.46	0.04	100.00	101.30	0.19	1.1
20	15	100.02	100.05	0.01	100.00	100.03	0.10	1.1
20	20	100.00	100.16	0.00	100.00	100.16	0.05	1.0
20	25	100.00	100.13	0.00	100.00	100.13	0.06	1.0
20	30	100.00	100.33	0.00	100.00	100.33	0.05	1.0
20	35	100.00	100.21	0.00	100.00	100.21	0.05	1.0
20	40	100.00	100.21	0.00	100.00	100.21	0.05	1.0
20	45	100.00	100.21	0.00	100.00	100.21	0.04	1.0
20	1,000	100.03	100.24	0.00	100.00	100.21	0.04	1.1
30	PDTSP	100.20	100.88	0.01	100.00	100.68	0.10	1.1
30	10	100.26	101.95	0.09	100.02	101.71	0.61	1.3
30	15	100.19	101.19	0.05	100.08	101.07	0.38	1.1
30	20	100.01	100.63	0.03	100.00	100.62	0.28	1.1
30	25	100.03	100.42	0.03	100.00	100.39	0.18	1.1
30	30	100.00	100.56	0.01	100.00	100.56	0.09	1.0
30	35	100.00	100.63	0.01	100.00	100.63	0.07	1.0
30	40	100.00	100.47	0.01	100.00	100.47	0.06	1.0
30	45	100.00	100.55	0.00	100.00	100.55	0.07	1.0
30	1,000	100.00	100.48	0.00	100.00	100.48	0.06	1.0
40	PDTSP	100.00	101.15	0.06	100.00	101.15	0.24	1.0
40	10	100.61	103.79	0.26	100.10	103.26	2.21	1.7
40	15	100.75	101.93	0.12	100.40	101.57	1.41	1.6
40	20	100.08	101.00	0.07	100.01	100.93	0.45	1.1
40	25	100.00	100.49	0.04	100.00	100.49	0.17	1.0
40	30	100.05	100.51	0.02	100.02	100.48	0.15	1.2
40	35	100.13	100.83	0.02	100.00	100.71	0.15	1.4
40	40	100.07	100.73	0.01	100.00	100.66	0.12	1.3
40	45	100.10	100.76	0.00	100.00	100.66	0.13	1.4
40	1,000	100.10	100.79	0.01	100.00	100.70	0.14	1.4
50	PDTSP	100.11	101.75	0.07	100.00	101.64	0.41	1.1
50	10	101.40	104.61	0.51	100.70	103.89	6.60	2.7
50	15	100.77	102.84	0.35	100.38	102.43	3.46	1.7
50	20	100.24	102.28	0.21	100.11	102.14	1.81	1.4
50	25	100.38	101.79	0.13	100.00	101.41	1.24	1.5
50	30	100.06	100.89	0.09	100.03	100.87	0.58	1.2
50	35	100.05	101.09	0.07	100.03	101.08	0.49	1.1
50	40	100.14	101.24	0.05	100.04	101.13	0.43	1.2
50	45	100.05	101.12	0.05	100.00	101.07	0.39	1.3
50	1,000	100.04	101.02	0.03	100.00	100.99	0.23	1.2
60	PDTSP	100.28	101.79	0.14	100.00	101.52	1.06	1.9
60	10	101.90	105.48	1.04	101.20	104.75	8.18	2.2
60	15	101.14	103.49	0.46	100.55	102.88	8.05	2.4
60	20	100.53	102.20	0.30	100.35	102.01	3.34	1.7
60	25	100.45	102.07	0.19	100.02	101.63	1.83	1.9
60	30	100.26	101.55	0.15	100.04	101.33	1.09	1.8
60	35	100.21	101.50	0.12	100.05	101.35	0.70	1.3
60	40	100.16	101.25	0.07	100.00	101.09	0.56	1.5
60	45	100.14	101.25	0.06	100.02	101.14	0.46	1.3
60	1,000	100.27	101.65	0.06	100.02	101.40	0.45	1.6

integer solution value *OPT* for the small instances, and the optimal value *TSP* of the TSP instance for the medium-large instances. The reason for computing the value *TSP* for large instances is to show how it relates to *LB* on our benchmark instances. The use of value *TSP* as lower bound is done in Gendreau et al. (1999) and, therefore, we also include it in our tables to facilitate comparison. Moreover, the relation between *TSP* and *OPT* also gives an idea of the added difficulty of solving a 1-PDTSP with respect to a TSP. The column headings have the following meanings:

n: is the number of vertices;

Q: is the capacity of the vehicle. Word "TSPPD" means that it is a TSPPD instance with the tightest capacity;

Table 3 Statistics on Medium/Large Instances of the 1-PDTSP and TSPPD

n	*Q*	*UB1/TSP*	*UB1/LB*	*UB1-t*	*UB2/TSP*	*UB2/LB*	*UB2-t*	*B&C*
100	PDTSP	101.29	101.58	0.71	101.10	101.39	2.23	1.7
100	10	166.48	107.98	5.79	164.41	106.65	29.38	3.1
100	15	135.65	108.59	4.92	134.48	107.69	22.85	2.4
100	20	121.46	107.50	2.95	120.81	106.92	18.74	2.6
100	25	112.42	104.81	1.66	112.10	104.50	13.81	2.4
100	30	108.68	104.07	1.13	107.41	102.86	9.10	3.1
100	35	105.18	102.25	0.73	104.79	101.87	5.11	2.1
100	40	103.89	102.02	0.54	103.38	101.51	3.14	2.2
100	45	102.51	101.83	0.40	102.09	101.42	2.92	2.2
100	1,000	100.39	101.18	0.13	100.05	100.84	1.04	2.2
200	PDTSP	101.81	102.40	4.33	101.22	101.80	16.49	3.0
200	10	183.76	113.23	32.18	181.91	112.09	151.96	3.4
200	15	151.23	115.72	32.46	149.16	114.10	195.39	3.9
200	20	131.47	113.77	21.21	129.96	112.44	123.89	3.2
200	25	121.26	111.08	12.75	119.58	109.54	121.31	3.9
200	30	115.20	109.42	8.23	113.46	107.77	103.73	4.0
200	35	110.39	106.82	5.87	109.21	105.68	70.02	3.4
200	40	107.40	105.37	4.09	106.70	104.69	51.30	3.0
200	45	105.50	104.31	3.07	104.58	103.40	43.99	3.8
200	1,000	101.12	101.91	0.49	100.29	101.08	6.39	3.5
300	PDTSP	102.27	102.84	8.29	101.21	101.78	53.05	5.0
300	10	189.33	119.55	96.05	188.23	118.86	285.56	2.7
300	15	154.42	120.95	97.88	152.36	119.34	395.63	3.3
300	20	136.55	119.70	66.09	134.87	118.23	364.38	3.3
300	25	125.40	116.30	40.00	124.23	115.22	209.55	2.3
300	30	118.13	113.08	26.54	116.66	111.66	250.58	3.5
300	35	112.97	109.99	18.40	111.30	108.37	214.15	4.3
300	40	110.20	108.51	13.83	108.55	106.88	194.84	4.7
300	45	107.97	107.10	10.66	106.64	105.78	209.06	5.0
300	1,000	101.47	102.30	1.09	100.72	101.54	26.34	5.0
400	PDTSP	102.42	103.04	17.33	101.49	102.10	118.63	5.7
400	10	182.90	120.27	179.50	181.40	119.28	453.64	3.2
400	15	150.54	121.74	195.05	149.27	120.71	585.37	2.7
400	20	133.36	118.75	129.53	131.42	117.01	423.05	3.1
400	25	123.56	115.61	77.57	121.78	113.93	396.96	4.0
400	30	117.19	112.80	48.18	115.69	111.35	280.79	2.8
400	35	113.00	110.56	32.30	111.44	109.03	281.56	4.1
400	40	109.26	107.93	24.41	108.07	106.76	290.91	4.8
400	45	107.41	106.70	18.79	105.84	105.15	294.42	6.0
400	1,000	101.63	102.37	1.87	100.69	101.43	38.69	5.6
500	PDTSP	102.51	103.25	19.55	101.60	102.34	130.10	5.8
500	10	187.41	126.03	341.24	186.24	125.25	708.95	2.9
500	15	153.29	126.15	357.77	151.94	125.04	913.39	3.1
500	20	136.73	122.23	232.69	135.50	121.13	836.03	4.0
500	25	126.10	118.69	148.57	124.81	117.48	435.31	2.2
500	30	118.84	114.65	96.39	118.17	114.01	581.24	3.3
500	35	114.80	112.56	70.58	113.70	111.48	330.60	3.1
500	40	111.42	110.29	49.46	110.26	109.13	350.61	3.6
500	45	108.90	108.37	37.39	107.82	107.29	228.92	3.3
500	1,000	101.80	102.67	3.08	100.95	101.82	70.78	6.0

UB1/OPT: (in Table 2) is the average percentage of the first heuristic over the optimal value for the small instances;

UB1/TSP: (in Table 3) is the average percentage of the first heuristic over the TSP optimal value for the large instances;

UB1/LB: is the average percentage of the first heuristic over the lower bound;

UB1-t: is the average CPU time (in seconds) of the first heuristic;

UB2/OPT: (in Table 2) is the average percentage of the incomplete optimisation algorithm over the optimal value for the small instances;

UB2/TSP: (in Table 3) is the average percentage of the incomplete optimisation algorithm over the TSP optimal value for the large instances;

UB2/LB: is the average percentage of the incomplete optimisation algorithm over the lower bound;

UB2-t: is the average CPU time (in seconds) of the incomplete optimisation algorithm (including the time of the first heuristic);

B&C: is the average number of calls to the branch-and-cut approach in the incomplete optimisation algorithm.

From Table 2, the basic heuristic described in §2 has a gap under 2% even on the hardest instances (i.e., $Q = 10$), and the computational effort was typically smaller than 1 second. Moreover, the worst average gap for the incomplete optimisation algorithm is 1% over the optimum, and the average time is under 1 minute. In general, the worst cases occur when the number of customers increase and the capacity decreases. If the capacity is large enough (i.e., $Q = 1{,}000$), the 1-PDTSP instances coincide with the TSP and the objective values computed by both heuristics are very close to the optimal TSP value. A similar behavior is observed for the TSPPD instances where the worst gap is 0.28% for the first heuristic algorithm, while the incomplete optimisation algorithm always found an optimal TSPPD solution.

When solving larger instances, it is not possible to compute the exact gap between the heuristic and the optimal value. From the instances where the optimum value has been calculated, one can conclude that *UB*2 and *UB*1 are nearer the optimum value *OPT* than the *LB*. The improvement of the incomplete optimisation algorithm over the first heuristic is about 1% on these instances. Both approximation procedures do not exceed 25% of the *LB*. The average computational time of the incomplete optimisation algorithm is close to 15 minutes for the hardest instances, which is a reasonable time for this type of routing problem. Again, as in Table 2, the problem is more difficult when Q decrease and when n increase, as expected. In all cases, the TSPPD instances are more difficult for the approaches than the TSP instances with the same value n, but they are much easier than the 1-PDTSP instances.

The quality of the first heuristic is worse when the 2-opt and the 3-opt procedures are deactivated. Indeed, the heuristic costs in respect to *LB* are between 105% and 120% in the small instances and between 120% and 190% in the medium-large instances, thus, proving that the first heuristic algorithm within only the multistart greedy scheme would guarantee worse results. The greedy algorithm takes about 1% of the total CPU time consumed by the first heuristic approach, i.e., most of the effort is dedicated to the 2-opt and 3-opt procedures.

The number of calls to the branch-and-cut algorithm in the incomplete optimisation algorithm was very small when $n \leq 60$, typically being 1 or 2. Hence, this means that the solution found by the initial heuristics tends to be also optimal of the defined neighborhood. Nevertheless, when $100 \leq n \leq 500$, the branch-and-cut code runs about four times, thus, improving the original solution with some additional effort. We have run the code using constraints (1)–(2) with different values of k, and value 5 seemed to provide the best compromise between solution quality and computational effort on our benchmark instances. Also, the choice of fixing to zero some x_e variables (i.e., the set definition of the set E' described in §3) produced the best results after experimenting with different options.

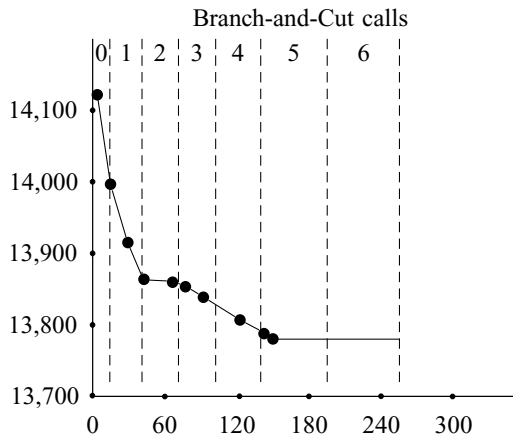

Figure 2 **Cost Value vs. Computational Time (Seconds) when Solving an Instance**

To illustrate the performance of the incomplete optimisation algorithm, Figure 2 shows the cost of the generated feasible solutions when solving a 1-PDTSP instance with $n = 200$ and $Q = 15$. Each black dot represents a solution improving a previous one. The vertical axis represents the solution cost and the horizontal axis the time when the solution was found. Each vertical dashed line separates consecutive iterations. Iteration 0 is the initial heuristic, where incumbent tour T^* was improved three times; only the best tour is plotted in the figure. The other iterations represent branch-and-cut executions, and iteration 6 is the last one where the last tour T^6 was not improved. It is observed that the use of the branch-and-cut code on the limited 1-PDTSP region defined by edges in E' and constraints (1)–(2) allows the incomplete optimisation algorithm to escape from local optimum in an effective way, mainly in the initial iterations. Figure 3 shows a table with more details of the best

h	0	1	2	3	4	5
cost value	14,125	13,915	13,861	13,839	13,807	13,780
time	3.5	30.4	67.2	93.3	123.9	151.4
$\lvert E^{h+1} \setminus E' \rvert$	0	2	1	3	3	4
$\lvert E^{h+1} \setminus E^h \rvert$	—	19	13	6	12	5

Figure 3 **Details of T^{h+1} Found at Iteration h when Solving an Instance**

tour found T^{h+1} at iteration h for each $h = 0, 1, \ldots, 5$. In particular, it shows the *cost value* of T^{h+1}, the *time* in seconds where it was found during iteration h, the number $|E^{h+1} \backslash E'|$ of edges in T^{h+1} not in E' (the set of edges associated with variables given to the branch-and-cut solver), and the number $|E^{h+1} \backslash E^h|$ of edges in T^{h+1} not in T^h. From the table, one observes that the primal heuristic can generate a new tour T^{h+1} using edges even outside E'.

We have generated 1,000 instances, from which 100 are TSPPD instances and 100 are TSP instances. Not one of the remaining 800 1-PDTSP instances resulted in being infeasible, because both our heuristic algorithms found feasible solutions, even when $Q = 10$. Moreover, we also imposed a time limit of 1,800 seconds to the overall execution of the incomplete optimisation algorithm, but this limit was never achieved, and no average time was bigger than 1,000 seconds.

Finally, to test the performance of our heuristic approaches solving TSPPD instances, we have run the computer codes of the instance generator and the tabu search heuristic used in Gendreau et al. (1999) on their three families of TSPPD instances. Tables 4 and 5 show the average percentages of gaps and times of their best heuristic algorithm (TS2) and our two heuristics (UB1 and UB2) solving the same instances on the same computer. Each table corresponds to the average results on the instances obtained modifying instances of the classical *vehicle routing problem* and the randomly generating Euclidean distance instances, respectively, we have also observed similar performances of our code on the randomly generated symmetric instances. An immediate observation is that our two heuristic approaches provide better results in less computational effort. A relevant remark from our experiments is that the TSPPD instances considered in Gendreau et al. (1999) are very close to the TSP instances, thus, they are easier to be solved than the 1-PDTSP instances with the same number of customers.

Table 4 Statistics on TSPPD Instances from Gendreau et al. (1999) Derived from VRP Test Instances

β	*TS2/TSP*	*TS2-t*	*UB1/TSP*	*UB1-t*	*UB2/TSP*	*UB2-t*	*B&C*
0	100.51	3.10	100.20	0.12	100.05	0.93	1.50
0.05	103.45	2.18	100.80	0.17	100.61	0.87	1.54
0.1	104.34	2.31	100.93	0.21	100.72	1.07	1.62
0.2	106.16	2.26	101.08	0.27	100.90	1.02	1.54

Table 5 Statistics on the Randomly Generated Euclidean TSPPD Instances from Gendreau et al. (1999)

β	n	*TS2/TSP*	*TS2-t*	*UB1/TSP*	*UB1-t*	*UB2/TSP*	*UB2-t*	*B&C*
0	25	100.00	0.16	100.04	0.06	100.00	0.07	1.1
	50	100.20	0.75	100.05	0.05	100.00	0.17	1.2
	75	100.78	1.82	100.20	0.11	100.00	0.44	1.5
	100	101.27	3.24	100.37	0.12	100.10	0.80	1.6
	150	102.37	8.35	100.81	0.24	100.34	2.55	2.7
	200	103.13	15.27	101.13	0.40	100.35	5.74	3.4
0.05	25	107.36	0.18	102.07	0.06	102.07	0.08	1.0
	50	103.95	0.60	100.20	0.06	100.17	0.19	1.2
	75	110.13	1.32	100.84	0.16	100.83	0.73	1.1
	100	107.23	2.37	100.61	0.14	100.39	0.97	1.9
	150	108.72	5.46	100.97	0.31	100.35	2.71	2.8
	200	108.57	11.15	101.31	0.53	100.69	5.74	3.0
0.1	25	108.21	0.17	101.22	0.04	101.18	0.10	1.1
	50	106.34	0.63	100.49	0.07	100.47	0.20	1.1
	75	113.28	1.42	101.42	0.22	101.32	1.00	1.4
	100	111.53	2.60	100.93	0.24	100.50	1.44	2.3
	150	110.90	6.19	101.26	0.50	100.68	3.52	3.0
	200	111.31	11.93	101.29	0.66	100.76	5.71	3.1
0.2	25	107.28	0.20	102.63	0.05	102.59	0.13	1.1
	50	106.53	0.72	100.80	0.09	100.79	0.25	1.1
	75	114.25	1.44	101.96	0.27	101.77	1.11	1.5
	100	113.09	2.61	101.35	0.67	100.96	1.94	2.0
	150	111.69	6.01	101.50	0.84	101.02	3.75	2.5
	200	113.28	11.85	101.74	1.01	100.99	8.54	3.5
∞	25	105.64	0.18	101.32	0.05	101.32	0.09	1.0
	50	110.86	0.73	102.50	0.09	102.47	0.58	1.2
	75	111.86	1.42	100.98	0.20	100.91	0.69	1.2
	100	110.34	2.59	101.14	0.25	100.87	1.45	1.7
	150	112.85	5.78	101.30	0.99	100.63	3.52	2.4
	200	113.03	11.21	101.56	1.39	100.83	10.50	3.6

Acknowledgments

This work was partially done while the second author was at the CRT, University of Montreal, by invitation of Gilbert Laporte and supported by the Canada Research Chair in Distribution Management. The authors thank Daniele Vigo for providing them with the instances and the FORTRAN codes used in Gendreau et al. (1999), and they thank three anonymous referees for their suggestions that improved an early version of this paper. This work was partially funded by Gobierno de Canarias (Research Project PI2000/116) and by Ministerio de Ciencia y Tecnología (Research Project TIC2000-1750-C06-02), Spain.

References

Anily, S., J. Bramel. 1999. Approximation algorithms for the capacitated traveling salesman problem with pickups and deliveries. *Naval Res. Logist.* **46** 654–670.

Anily, S., R. Hassin. 1992. The swapping problem. *Networks* **22** 419–433.

Anily, S., G. Mosheiov. 1994. The traveling salesman problem with delivery and backhauls. *Oper. Res. Lett.* **16** 11–18.

Anily, S., M. Gendreau, G. Laporte. 1999. The swapping problem on a line. *SIAM J. Comput.* **29** 327–335.

Baldacci, R., A. Mingozzi, E. Hadjiconstantinou. 2003. An exact algorithm for the traveling salesman problem with mixed delivery and collection constraints. *Networks* **42** 26–41.

Bianco, L., A. Mingozzi, S. Riccardelli, M. Spadoni. 1994. Exact and heuristic procedures for the traveling salesman problem with precedence constraints, based on dynamic programming. *INFOR* **32** 19–32.

Chalasani, P., R. Motwani. 1999. Approximating capacitated routing and delivery problems. *SIAM J. Comput.* **28** 2133–2149.

Fischetti, M., A. Lodi. 2003. Local branching. *Math. Programming* **98** 23–47.

Frederickson, G. N., M. S. Hecht, C. E. Kim. 1978. Approximation algorithms for some routing problems. *SIAM J. Comput.* **7** 178–193.

Garey, M. R., D. S. Johnson. 1979. *Computers and Intractability: A Guide to the Theory of $\mathcal{NP}$-Completeness*. W. H. Freeman and Company, San Francisco, CA.

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

Gendreau, M., G. Laporte, D. Vigo. 1999. Heuristics for the traveling salesman problem with pickup and delivery. *Comput. Oper. Res.* **26** 699–714.

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

Healy, P., R. Moll. 1995. A new extension of local search applied to the dial-a-ride problem. *Eur. J. Oper. Res.* **83** 83–104.

Helsgaun, K. 2000. An effective implementation of the Lin-Kernighan traveling salesman heuristic. *Eur. J. Oper. Res.* **12** 106–130.

Hernández-Pérez, H., J. J. Salazar-González. 2004. A branch-and-cut algorithm for the traveling salesman problem with pickup and delivery. *Discrete Appl. Math.* **144**.

Johnson, D. S., L. A. McGeoch. 1997. The traveling salesman problem: A case study in local optimization. E. J. L. Aarts, J. K. Lenstra, eds. *Local Search in Combinatorial Optimization*. John Wiley and Sons, Chichester, U.K.

Kalantari, B., A. V. Hill, S. R. Arora. 1985. An algorithm for the traveling salesman problem with pickup and delivery customers. *Eur. J. Oper. Res.* **22** 377–386.

Kubo, M., H. Kasugai. 1990. Heuristic algorithms for the single vehicle dial-a-ride problem. *J. Oper. Res. Soc. Japan* **33** 354–364.

Lin, S. 1965. Computer solutions of the traveling salesman problem. *Bell System Tech. J.* **44** 2245–2269.

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

Mosheiov, G. 1994. The travelling salesman problem with pick-up and delivery. *Eur. J. Oper. Res.* **79** 299–310.

Psaraftis, H. 1983. Analysis of an $O(n^2)$ heuristic for the single vehicle many-to-many Euclidean dial-a-ride problem. *Transportation Res.* **17** 133–145.

Renaud, J., F. F. Boctor, G. Laporte. 2002. Perturbation heuristics for the pickup and delivery traveling salesman problem. *Comput. Oper. Res.* **29** 1129–1141.

Renaud, J., F. F. Boctor, I. Ouenniche. 2000. A heuristic for the pickup and delivery traveling salesman problem. *Comput. Oper. Res.* **27** 905–916.

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