A hybrid heuristic approach for the multi-commodity pickup-and-delivery traveling salesman problem☆

Hipólito Hernández-Pérez[a], Inmaculada Rodríguez-Martín[a], Juan-José Salazar-González[a]

[a]*DMEIO, Facultad de Ciencias, Universidad de La Laguna, 38271 La Laguna, Tenerife, Spain*

Abstract

We address in this article the multi-commodity pickup-and-delivery traveling salesman problem, which is a routing problem for a capacitated vehicle that has to serve a set of customers that provide or require certain amounts of m different products. Each customer must be visited exactly once by the vehicle, and it is assumed that a unit of a product collected from a customer can be supplied to any other customer that requires that product. Each product is allowed to have several sources and several destinations. The objective is to minimize the total travel distance. We propose a hybrid three-stage heuristic approach that combines a procedure to generate initial solutions with several local search operators and shaking procedures, one of them based on solving an integer programming model. Extensive computational experiments on randomly generated instances with up to 400 locations and 5 products show the effectiveness of the approach.

Keywords: Pickup-and-delivery; hybrid approach; traveling salesman; local search.

1. Introduction

The *Multi-Commodity Pickup-and-Delivery Traveling Salesman Problem* (m-PDTSP) is a generalization of the *Traveling Salesman Problem* in which a capacitated vehicle based at

☆This work was supported by the research project MTM2012-36163-C06-01, "Ministerio de Economía y Competitividad", Spain.

Email addresses: hhperez@ull.edu.es (Hipólito Hernández-Pérez), irguez@ull.es (Inmaculada Rodríguez-Martín), jjsalaza@ull.es (Juan-José Salazar-González)

a *depot* must visit a set of *customers*. Each location (depot and customers) must be visited exactly once by the vehicle. The travel cost between each pair of locations is known, and not necessarily symmetric. Each customer requires some given quantities of different products and/or provides some given quantities of other different products. A unit of a product collected from a customer can be supplied to any customer that requires this product. It is assumed that the vehicle must start and finish the route at the depot. Another visit to the depot is not allowed. The aim of the m-PDTSP is to find a Hamiltonian route for the vehicle such that it picks up and delivers all the quantities of the different products satisfying the vehicle-capacity limitation and minimizing the total travel cost. Since each customer is visited once, each unit of a product loaded on the vehicle stays on the vehicle until it is delivered. For that reason we say that the m-PDTSP is a *non-preemptive* problem.

The initial load of any product in the vehicle when leaving the depot is unknown, and must be determined within the optimization problem. The variant of the m-PDTSP where the initial load of any product is fixed can also be solved through the approach described in this paper with a slight modification of the instance. Note that, the initial load being unfixed, the vehicle is allowed to deliver a demand immediately when leaving the depot and collect the associated product afterwards. In other words, there are not a priori precedence relations between pickup and delivery locations of a commodity in the route. Still, the approach described in this paper can be adapted to the variant where the the initial load is required to be zero. This assumption is argued in [16].

An application of the m-PDTSP occurs in the context of inventory repositioning, as pointed out by Anily and Bramel [1]. Assume that a set of retailers is geographically dispersed in a region. Often, due to the random nature of the demand, some retailers have an excess of inventory of some products while others need additional stock. In many cases the company may decide to transfer inventory from retailers with low sales to those with high sales. Determining the cheapest way to execute a given stock transfer (with the requirement that each location has to be visited exactly once) is the m-PDTSP.

Another application arises in the context of a self-service bike hiring system, where every night a capacitated vehicle must visit the bike stops in a city to collect or deliver bikes to

restore the initial configuration of the system. Chemla et al. [5] and Raviv et al. [23], among others, approached the case where the bikes are all identical (i.e., one product) as a 1-PDTSP. When there are different types of bikes (e.g., with and without baby chairs) the problem can be described as the m-PDTSP.

The m-PDTSP is $\mathcal{NP}$-hard in the strong sense since it coincides with the TSP when the vehicle capacity is large enough. What is more, even checking the existence of a feasible solution for an instance with $m = 1$ is a strongly $\mathcal{NP}$-complete problem (see [13]). The m-PDTSP was introduced by Hernández-Pérez and Salazar-González [16]. They proposed a Mixed Integer Linear Programming model for the m-PDTSP, and described a branch-and-cut algorithm able to solve instances with up to 30 customers and 3 commodities. Since exact algorithms are only able to cope with small instances, heuristic approaches are needed to tackle larger instances in practice. This is the main motivation of our paper.

A closely related problem to the m-PDTSP is the *Non-Preemptive Capacitated Swapping Problem* (NCSP), proposed by Erdoğan et al. [6]. As in the m-PDTSP, the NCSP considers one depot, a set of customers, and several commodities with many sources and many destinations. In the NCSP, however, each customer concerns one commodity (either as pickup or delivery location) or two commodities (one as pickup location and another as delivery location). In addition to customer locations, the NCSP also includes transshipment locations, where some commodities can be temporarily dropped off. Customer and transshipment locations may be visited zero, one or two times by the vehicle, while the depot may be visited up to three times. The demand of a customer cannot be split and a commodity cannot be dropped off in an intermediate customer. The NCSP consists of finding a minimum-cost route satisfying all customer's requests. Erdoğan et al. [6] describe a branch-and-cut algorithm to solve instances with up to 20 locations and 8 commodities, and 30 locations and 4 commodities. Bordenave et al. [3] present a branch-and-cut algorithm to solve the particular case of the NCSP where the vehicle capacity and customer demands are all equal to one. They solved instances with up to 200 locations and 8 commodities.

The *one-to-one m-PDTSP* is a particular case of the m-PDTSP where each commodity has one origin and one destination. It can be considered as a Dial-a-Ride routing problem

without time window requirements. The one-to-one m-PDTSP assumes that the load of the vehicle when leaving the depot is zero, unless the depot is the source of a commodity. Hernández-Pérez and Salazar-González [15] describe a branch-and-cut algorithm for this problem solving instances involving up to 24 customers and 15 commodities. Rodríguez-Martín and Salazar-González [24] propose and compare several metaheuristic approaches to solve instances with up to 300 customers and 600 commodities.

Some articles dealing with one-commodity variants are the following. Chalasani and Motwani [4] study the special case of the 1-PDTSP where the delivery and pickup demands are all equal to one. This problem is called *Q-delivery TSP*, where Q is the capacity of the vehicle. Anily and Bramel [1] consider the same problem with the name *Capacitated TSP with Pickups and Deliveries*. Hernández-Pérez and Salazar-González [14] present an exact algorithm for the 1-PDTSP solving instances with up to 200 customers. Hernández-Pérez et al. [12] describe a hybrid algorithm that combines Greedy Randomized Adaptive Search Procedure and Variable Neighborhood Descent paradigms. Zhao et al. [26] propose a Genetic Algorithm that on average gives better results. Finally, Mladenović et al. [17] describe a General Variable Neighborhood Search improving the best-known solution for all benchmark instances and solving instances with up to 1000 customers.

As in most of the articles dealing with TSP variants, we have decided to keep the assumption that each customer in the m-PDTSP must be visited once. However, the literature on vehicle routing includes articles that do not make this assumption. In particular, some authors address variants where a customer must be visited at most once (e.g. [7, 8]), or where a customer must be visited at least once (e.g. [2, 19, 20, 25]). There are also articles on related problems considering more than one vehicle, as in Psaraftis [22], but, to our knowledge, in all of them each commodity must be transported from one source to one destination. Our paper is the first one proposing an approach for dealing with large instances of a capacitated pickup-and-delivery problem where each commodity may have several sources and several destinations.

The rest of the paper is organized as follows. Section 2 gives the formal definition of the problem and presents the notation used throughout the paper. Section 3 describes

the heuristic algorithm to solve the m-PDTSP. Section 4 is devoted to the tuning of the algorithm's parameters. Computational results are shown in Section 5, and final remarks are made in Section 6.

2. Problem definition and notation

The m-PDTSP is defined on a complete directed graph $G = (V, A)$. The vertex set $V = \{1, \ldots, n\}$ represents the locations. Vertex 1 is the depot and can be identified in the rest of the paper as a customer. For each pair of customers i and j we have the arc $a = (i, j) \in A$ and a travel cost c_{ij}. Let $K = \{1, \ldots, m\}$ be the set of products. For each customer $i \in V$ and each product $k \in K$ let q_i^k be the demand of product k associated with i. When $q_i^k > 0$ customer i provides q_i^k units of product k and when $q_i^k < 0$ customer i requires $-q_i^k$ units of product k. We assume that $\sum_{i \in V} q_i^k = 0$ for all $k \in K$, i.e., each product is conserved through the route. The capacity of the vehicle is denoted by Q.

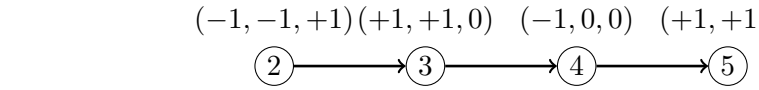
As mentioned above, contrary to what happens in the TSP, finding a feasible solution (optimal or not) for the m-PDTSP is a problem with a hard computational complexity. Nevertheless, checking if a given TSP solution is feasible for m-PDTSP can be done in $O(mn)$ time. Indeed, let us consider a path $\overrightarrow{P}$ defined by the vertex sequence $v_1, \ldots, v_s$ for $s \leq n$. Let $l_i^k(\overrightarrow{P}) := l_{i-1}^k(\overrightarrow{P}) + q_{v_i}^k$ be the load of the vehicle when coming out from v_i, considering that the vehicle enters customer v_1 with load $l_0^k(\overrightarrow{P})$. Notice that $l_i^k(\overrightarrow{P})$ could be negative if $l_0^k(\overrightarrow{P}) = 0$ and, therefore, the minimum quantity of load of commodity k for a feasible solution through the path $\overrightarrow{P}$ is $-\min_{i=0}^{s}\{l_i^k(\overrightarrow{P})\}$. With this notation, $\overrightarrow{P}$ is an infeasible path if

$$\max_{i=0}^{s}\left\{\sum_{k \in K} l_i^k(\overrightarrow{P})\right\} - \sum_{k \in K} \min_{i=0}^{s}\left\{l_i^k(\overrightarrow{P})\right\} > Q. \tag{1}$$

For example, consider an m-PDTSP instance with $m = Q = 3$ and customers 2, 3, 4 and 5 with demand vectors $(-1, -1, +1)$, $(+1, +1, 0)$, $(-1, 0, 0)$ and $(+1, +1, 0)$, respectively. The path defined by the customer sequence 2, 3, 4 and 5 is infeasible because

$$\max_{i=0}^{s}\left\{\sum_{k=1}^{m} l_i^k(\overrightarrow{P})\right\} - \sum_{k=1}^{m} \min_{i=0}^{s}\left\{l_i^k(\overrightarrow{P})\right\} = 4 > Q.$$

Figure 1: Infeasible path when $m = 3$ and $Q = 3$.



$l^1_*(\overrightarrow{P}) =$	0	$\boxed{-1}$	0	-1	0
$l^2_*(\overrightarrow{P}) =$	0	$\boxed{-1}$	0	0	$+1$
$l^3_*(\overrightarrow{P}) =$	$\boxed{0}$	$+1$	$+1$	$+1$	$+1$
$\sum_k l^k_*(\overrightarrow{P}) =$	0	-1	$+1$	0	$\boxed{+2}$

Figure 1 illustrates the calculations done in this example.

It is easy to see that the reverse path $\overleftarrow{P}$ of $\overrightarrow{P}$, defined by the vertex sequence $v_s, \ldots, v_1$, is infeasible if

$$\sum_{k \in K} \max_{i=0}^{s} \left\{ l^k_i(\overrightarrow{P}) \right\} - \min_{i=0}^{s} \left\{ \sum_{k \in K} l^k_i(\overrightarrow{P}) \right\} > Q.$$

Then $\overrightarrow{P}$ can be feasible and $\overleftarrow{P}$ infeasible, and vice versa. For example, $\overrightarrow{P}$ is infeasible and $\overleftarrow{P}$ is feasible in Figure 1.

We define the *infeasibility* of the path $\overrightarrow{P}$ as:

$$\mathit{infeas}(\overrightarrow{P}) := \max \left\{ 0, \max_{i=0}^{s} \left\{ \sum_{k \in K} l^k_i(\overrightarrow{P}) \right\} - \sum_{k \in K} \min_{i=0}^{s} \left\{ l^k_i(\overrightarrow{P}) \right\} - Q \right\}.$$

A path is feasible when $\mathit{infeas}(\overrightarrow{P}) = 0$.

Note that the infeasibility of any path $\overrightarrow{P}$ does not depend on the initial loads $l^k_0(\overrightarrow{P})$ for $k = 1, \ldots, m$. For example, if $l^1_0(\overrightarrow{P})$ is increased by one unit, then $l^1_i(\overrightarrow{P})$ for all $i = 1, \ldots, s$, $\max^s_{i=0} \left\{ \sum_{k \in K} l^k_i(\overrightarrow{P}) \right\}$, $\min^s_{i=0} \left\{ l^1_i(\overrightarrow{P}) \right\}$, and $\sum_{k \in K} \min^s_{i=0} \left\{ l^k_i(\overrightarrow{P}) \right\}$ increase all by one too. Therefore the value $\mathit{infeas}(\overrightarrow{P})$ remains the same.

We end this section with a mathematical formulation for the m-PDTSP. Let us consider a 0-1 variable x_a for each arc $a \in A$ assuming value 1 if and only if a is routed by the vehicle. For a subset $S \subset V$ we denote by $\delta^+(S)$ the set of arcs $\{(i, j) : i \in S \text{ and } j \in V \setminus S\}$ and $\delta^-(S)$ the set of arcs $\{(i, j) : i \in V \setminus S \text{ and } j \in S\}$. Finally, for a subset $A' \subseteq A$ we write

$x(A')$ instead of $\sum_{a \in A'} x_a$. Then the m-PDTSP can be formulated as:

$$\min \sum_{a \in A} c_a x_a \tag{2}$$

subject to

$$x(\delta^-(\{i\})) = 1 \qquad \text{for all } i \in V \tag{3}$$

$$x(\delta^+(\{i\})) = 1 \qquad \text{for all } i \in V \tag{4}$$

$$x(\delta^+(S)) \geq 1 \qquad \text{for all } S \subset V \tag{5}$$

$$x_a \in \{0,1\} \qquad \text{for all } a \in A \tag{6}$$

$$x\left(\bigcup_{k \in K} \delta^+(S^k)\right) \geq \left\lceil \frac{1}{Q} \sum_{k \in K} \sum_{i \in S^k} q_i^k \right\rceil \qquad \text{for all } S^k \subset V, \;\; k = 1, \ldots, m. \tag{7}$$

Constraints (3)–(7) guarantee that x is a Hamiltonian tour. Inequalities (7) are valid for feasible paths and eliminate infeasible paths, as observed in [16].

3. The Algorithm for the m-PDTSP

To solve the m-PDTSP we propose a three-stage algorithm. The first stage (initialization phase) consists of a procedure that generates a set of initial (not necessarily feasible) solutions. In the second stage (improvement phase), a Variable Neighborhood Descent (VND) procedure is applied to each of the solutions selected in the first stage. Six operators are proposed to compose this VND. Finally, in the third stage (perturbation and refinement phase), each local minimum is perturbed applying a combination of three shaking procedures. The second and third stages are repeated in the hope of further improvements. Each stage is subject to a stopping criterium, and the best feasible solution found during the whole algorithm is given as the final solution.

The whole scheme can be considered as a hybrid heuristic algorithm since it combines the Multi-Start and the VND metaheuristic paradigms, and it also makes use of Mathematical Programming techniques to perform local search. We describe next in detail each of the components.

3.1. Initialization phase

This phase creates a set of initial solutions. To generate a (not necessarily feasible) solution we apply a randomized greedy procedure based on the *nearest neighbor heuristic* approach for the TSP. It extends a partial path starting from a random vertex and iteratively adding a new location until they are all in the path. At each iteration a customer is added at the end of the partial path, trying to obtain a feasible solution of good quality. To this end, we evaluate all possible candidates and introduce a vertex chosen randomly among the nearest three to the last vertex in the path that keeps the partial solution feasible. More precisely, if v_1, v_2 and v_3 are the nearest vertices to the last vertex added that keep the partial tour feasible, we choose v_i $(i = 1, 2, 3)$ with probability α_i $(\alpha_1 + \alpha_2 + \alpha_3 = 1)$. If it is not possible to have feasibility, we introduce the vertex that minimizes the infeasibility.

Another feature of this initial phase is that the travel costs are modified trying to obtain a feasible solution (or minimizing the infeasibility of the solution). The distances d_{ij} considered are given by:

$$d_{ij} = c_{ij} + C \cdot \frac{\sum_{a \in A} c_a}{|A|} \cdot \frac{2Q - \sum_{k=1}^{m} (|q_i^k| + |q_j^k|)}{2Q}$$

for a parameter C. The constant $\sum_{a \in A} c_a / |A|$ is used to normalize each instance. The aim of considering this cost modification is to favor a route going from a customer i to a customer j with large (absolute) demand. Similar ideas are presented in Hernández-Pérez and Salazar-González [13], Zhao et al. [26] and Mladenović et al. [17] for the 1-PDTSP.

The procedure is performed MS_0 times in the first phase of the algorithm to generate a set of initial solutions. A hash structure is used to avoid saving a route several times in the set. Then the MS_1 solutions with the smallest infeasibility and cost are taken as input data for the second phase, next described.

3.2. Improvement phase

In the second stage of our algorithm, a VND procedure starts from each of the solutions selected in the first stage. VND is a variant of the well-known Variable Neighborhood Search technique in which several local search operators are applied, one after the other, to explore

different neighborhoods (see [10]). Inspired by the works of Mladenović et al. [17] for the 1-PDTSP and Rodríguez-Martín and Salazar-González [24] for the one-to-one m-PDTSP, we propose six operators to explore neighbourhoods of a solution. The best combinations of these operators are described in Section 4.2.

The 2-opt move is the simplest k-opt edge-exchange method. It consists of removing two arcs (v_i, v_{i+1}) and (v_j, v_{j+1}) from the tour and reconnecting the two paths created by adding (v_i, v_j) and (v_{i+1}, v_{j+1}). This implies reversing the path from v_j to v_{i+1}, which may decrease or increase the cost of the new solution. We also consider the tour in the reverse orientation, and the whole operator is denoted by `2-opt`. To evaluate the convenience of a move, we first evaluate the infeasibility and then the cost of the solutions. That is, a 2-opt move is performed only if the new solution is less infeasible, or if it has the same infeasibility but is shorter.

It is known for the TSP (see, e.g., Hansen and Mladenović [11]) that it is not necessary to explore the $n(n-1)/2$ possible 2-opt moves because, to have a cost reduction, at least one of the added arcs must be shorter than one of the removed arcs. Therefore, to get an efficient implementation, for each vertex v_i we first compute a sorted list of its neighbors according to increasing distance from v_i. Then, for each arc (v_i, v_{i+1}) to remove, we select the arc (v_j, v_{j+1}) such that

$$c_{v_i v_{i+1}} + c_{v_j v_{j+1}} > c_{v_i v_j} + c_{v_{i+1} v_{j+1}},$$

taking v_j from the sorted list for v_i until $c_{v_i v_j} \geq c_{v_i v_{i+1}}$.

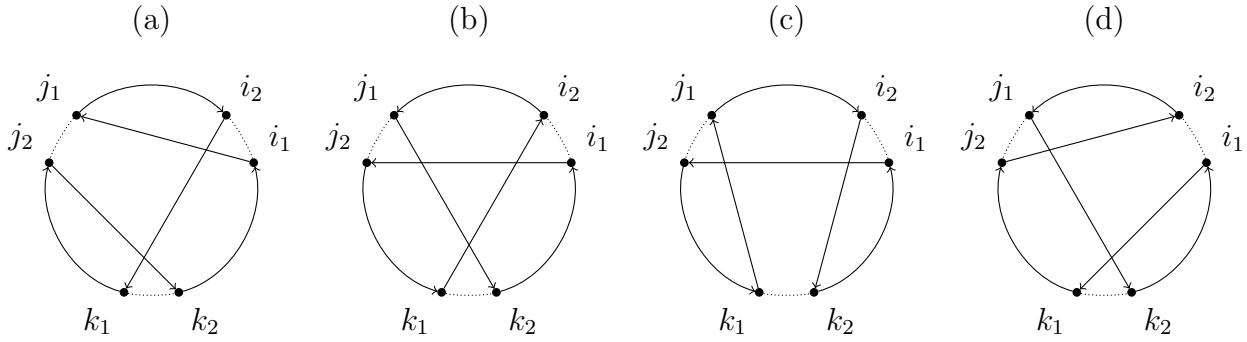
However, if the original tour is infeasible (that is, if $\mathit{infeas}(\overrightarrow{T}) > 0$) we allow the exploration of longer solutions. To this end we relax the above condition as follows. We introduce a parameter M and allow the algorithm checking all possible 2-opt moves satisfying

$$c_{v_i v_{i+1}} + c_{v_j v_{j+1}} + M > c_{v_i v_j} + c_{v_{i+1} v_{j+1}}.$$

Vertices v_j from the sorted list are processed until $c_{v_i v_j} + M \geq c_{v_i v_{i+1}}$.

The 3-opt move consists of removing three arcs from the solution and reconnecting the paths by adding three arcs. There are four types of 3-opt moves to explore (excluding

Figure 2: 3-opt moves.



degenerate cases like 2-opt moves). They are illustrated in Figure 2. Note that move (b) keeps the orientation of the three paths, while the other three moves require changing the orientation of some paths. As confirmed by our experiments (see Section 4.2), move (b) is likely to keep feasibility of a tour, and for that reason is more adapted for an improvement phase rather than for the perturbation phase. The operator exploring moves of type (b) is denoted by `3-opt-b`, the operator exploring the other three moves by `3-opt-acd`, and the operator exploring all the four 3-opt moves by `3-opt-abcd`.

Insertion is a particular case of a 3-opt move where two of the three removed arcs are consecutive. Then, the effect of reconnecting the tour is that of moving a vertex forward or backward in a sequence. The forward insertion tries to find a shorter tour by moving a customer from its current position i 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. This operator is denoted by `fw-ins`. The backward insertion 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. The operator is denoted by `bw-ins`.

Before making a k-opt move, it is necessary to analyze the feasibility of the new tour. This implies evaluating equation (1), which requires $O(mn)$ time when each term of the equation is computed. However, it can be done in a more efficient way using *binary indexed*

tree (BIT) data structures as proposed in Mladenović et al. [17] for the 1-PDTSP. The BIT structure can be used to calculate the maximum (or minimum) of an array of n elements in a time $O(\log n)$. This allows to reduce the complexity of the feasibility checking of a m-PDSTP solution to $O(m \log n)$.

Briefly, let $\overrightarrow{T} = (v_1, \ldots, v_n)$ be a tour and let $l_i^k(\overrightarrow{T})$ be the amount of product k in the vehicle when leaving v_i $(i = 1, \ldots, n)$. A BIT structure is built for each $k \in K$, storing the values $l_i^k(\overrightarrow{T})$ in the leaves of a the tree. If n is not a power of 2, the structure can be completed with leaves storing 0. Each parent node in the BIT stores the minimum of the quantities stored in its two children nodes. The construction of the BIT can be done in $O(n)$ time for each k. Similarly, an extra BIT structure is used to calculate the maximum of the values $\sum_{k \in K} l_i^k$. Mladenović et al. [17] give a pseudo-code and a detailed example of this procedure for the 1-PDTSP which can be easily adapted for the m-PDTSP.

As observed in Hansen and Mladenović [11], one can analyze all solutions of a k-opt operator and select the best one, or quit the operator with the first solution that improves the initial tour. Based on preliminary experiments, we opted for the strategy that analyzes all solutions.

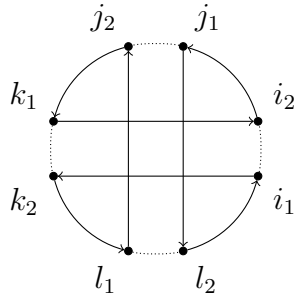
A related approach for the dial-a-ride routing problem was originally introduced by Psaraftis [21]. This problem is a particular case of the uncapacitated one-to-one m-PDTSP, and the approach in [21] has not an immediate extension to the m-PDTSP.

3.3. Perturbation and refinement phase

To escape from local minima, we make use of *shaking procedures*, which are random moves to perturb solutions. We now describe three shaking procedures. Different combinations of them are empirically analyzed in Section 4.3.

The first shaking procedure is denoted by `3-opt-rand` and performs a random 3-opt move, i.e. it selects three non-consecutive arcs and reconstructs a new tour according to Figure 2. If the infeasibility of the new tour is bigger than the one of the initial tour, another random move is done. On the other hand, a new tour with worse cost is allowed as long as the infeasibility does not increase. In the worse case, all the 3-opt moves could be examined.

Figure 3: Double-bridge move.



The second shaking procedure is called *double-bridge* and denoted by `db-rand`. It performs random 4-opt moves keeping the orientation of the four paths. The type of move is illustrated in Figure 3. Other types of 4-opt moves were considered but discarded because checking the feasibility issue is quite time consuming and reversing paths rarely improves the feasibility with respect to the initial tour. As before, `db-rand` is repeated until the infeasibility of the new tour is equal to or smaller than that of the initial tour.

The third shaking procedure, denoted by `mip-rand`, uses the mathematical formulation (2)–(7). To avoid solving a large and complex model, a given percentage of binary variables are a priori fixed, thus reducing the size of the models. This approach is similar to other variable fixing approaches in the literature like the so-called kernel search (see, e.g., Guastaroba and Speranza [9]). In our implementation, the variables fixed to 1 are chosen randomly among those corresponding to arcs used in the incumbent solution $\overrightarrow{T}$. The restricted model is then solved using the branch-and-cut algorithm described in [16]. To speed up the algorithm, a portion of variables with large modified travel cost d_{ij} are fixed to 0 also, unless it corresponds to an arc in $\overrightarrow{T}$. The percentages of binary variables fixed to 1 and fixed to 0 are dynamically set in order to keep the optimality gap small. The dynamic setting is explained in Section 4.

The branch-and-cut algorithm is initialized with the current best solution. In addition, it includes a primal heuristic procedure based on the nearest neighborhood for the TSP. More precisely, given a fractional solution x^*, it uses the approach described in Section 3.1 with

Table 1: Computational results to decide α and C in the initialization phase.

	$\alpha = (0.90, 0.05, 0.05)$														
	$C = 0$			$C = 1$			$C = 2$			$C = 5$			$C = 10$		
Name	fea	avg	rep	fea	avg	rep	fea	avg	rep	fea	avg	rep	fea	avg	rep
m4n100Q40A	0.0	–	0.0	0.3	39.8	0.0	0.8	35.0	0.0	6.1	45.0	0.0	14.8	68.6	0.0
m4n100Q60A	30.5	60.2	0.0	38.3	59.3	0.0	46.9	63.3	0.0	52.5	79.7	0.0	52.4	108.6	0.0
m4n200Q40A	0.0	–	0.0	0.0	–	0.0	0.0	–	0.0	3.3	30.1	0.0	28.2	52.5	0.0
m4n200Q60A	23.1	37.7	0.0	32.2	38.9	0.0	39.1	41.7	0.0	67.2	56.8	0.0	85.9	80.5	0.0
m4n400Q40A	0.0	–	0.0	0.0	–	0.0	0.0	–	0.0	3.1	26.7	0.0	17.5	52.0	0.0
m4n400Q60A	19.9	33.2	0.0	25.9	33.6	0.0	39.6	38.8	0.0	59.0	59.3	0.0	63.2	89.3	0.0
Avg.	12.3	45.8	0.0	16.1	45.5	0.0	21.1	48.7	0.0	31.9	62.5	0.0	43.7	82.7	0.0
	$\alpha = (0.98, 0.01, 0.01)$														
	$C = 0$			$C = 1$			$C = 2$			$C = 5$			$C = 10$		
Name	fea	avg	rep	fea	avg	rep	fea	avg	rep	fea	avg	rep	fea	avg	rep.
m4n100Q40A	0.0	–	14.7	0.0	–	14.1	0.6	25.8	11.8	7.1	39.9	9.1	14.0	64.6	12.0
m4n100Q60A	29.0	53.5	8.6	37.8	53.5	7.9	44.5	58.2	7.4	49.1	74.7	7.1	50.4	101.7	7.3
m4n200Q40A	0.0	–	0.9	0.0	–	1.4	0.1	12.6	0.5	3.5	28.3	0.2	33.3	48.3	0.2
m4n200Q60A	22.6	33.4	0.1	31.0	33.7	0.1	37.7	37.4	0.6	64.1	52.7	0.3	87.3	76.4	0.2
m4n400Q40A	0.0	–	0.4	0.0	–	0.1	0.1	8.9	0.0	3.4	23.1	0.0	18.5	48.8	0.0
m4n400Q60A	17.9	28.9	0.0	28.9	29.7	0.1	40.6	35.2	0.0	59.8	54.7	0.0	64.4	84.4	0.0
Avg.	11.6	40.6	4.1	16.3	40.2	4.0	20.6	44.1	3.4	31.2	57.6	2.8	44.7	77.1	3.3

$d_{ij} = 1 - x^*_{ij}$ for each arc (i, j). The procedure is repeated starting from different vertices, and the 2-opt local search operator described in Section 3.2 is applied to the best solution if feasible. The branch-and-cut algorithm stops if a time limit is reached or the number of branching nodes exceeds a given threshold.

4. Tuning of the parameters

To calibrate the algorithm and find reasonable values for its different parameters, we conducted preliminary computational experiments using a set of instances with $m = 4$, $n \in \{40, 60, 80, 100, 200, 400\}$ and $Q \in \{40, 60\}$. The way these instances were generated is explained in Section 5. We considered one instance for each n and Q, and the execution of the algorithm was limited to n seconds on each instance. We will next comment the calibration of the parameters involved in the algorithm.

4.1. Initialization phase

To tune the parameters C and α of the initial phase, we run it 1000 times over each of the medium and large test instances ($n \in \{100, 200, 400\}$). The computational time was less than 0.01 seconds for each run. Table 1 shows the percentage of feasible solutions obtained (fea), the average percentage of deviation between the best feasible solution found and the best known solution (avg), and the percentage of repeated solutions (rep) for different values of α and C.

We can see that the value of α does not affect the percentage of feasible solutions much, but it has a great influence in the quality of the solutions. Larger α_1 values produce best quality feasible solutions, but also a higher percentage of repeated solutions, especially for small sizes n. On the other hand, the larger the C value, the larger the percentage of feasible solutions and the smaller the quality of those solutions. Note that the algorithm does not find any feasible solution when $Q = 40$ and $C = 0$. Taken all this into account, we decided to set $\alpha = (1 - \frac{2}{n}, \frac{1}{n}, \frac{1}{n})$ and C to a variable value initialized to $C = 2$ which is increased when the rate of feasible solutions is low and is decreased when the rate of feasible solutions is high. More precisely, when the rate (observed in the last 1000 runs of the initial heuristic) is smaller than 0.5% the parameter C is multiplied by 1.2, and when the rate is greater than 20% the parameter C is multiplied by 0.9.

The third parameter in the initialization phase is MS_0, the number of times that the randomized greedy procedure is launched. The fourth parameter is MS_1, the number of solutions selected from the initialization phase. These parameters affect the improvement phase. For that reason we conducted experiments by running the 2-opt operator with different values for MS_0 and MS_1. Based on our experiments we decided to set $MS_0 = 200000$ and $MS_1 = 20$. At first glance, the high value of MS_0 may be surprising. This decision is explained by three arguments. First, the randomized greedy procedure is quite simple and fast. Second, the improvement operators hardly produce feasible solutions from infeasible ones. More precisely, in our preliminary experiments only 12% of infeasible greedy solutions yield feasible solutions through the 2-opt operator. Third, we observed a high correlation coefficient ($\rho = 0.74$ in our experiments) between the solution value from the

Table 2: Computational results to select the operators for the improvement phase.

	2-opt fw-ins bw-ins		2-opt fw-ins bw-ins 3-opt-abcd		2-opt fw-ins bw-ins 3-opt-b		2-opt fw-ins bw-ins 3-opt-b 3-opt-acd	
Name	avg	time	avg	time	avg	time	avg	time
m4n100Q40A	16.2	0.2	15.4	0.3	15.8	0.2	15.4	0.3
m4n100Q60A	21.5	0.1	19.8	0.2	20.7	0.1	19.9	0.2
m4n200Q40A	11.5	0.5	10.6	2.4	11.0	0.8	10.6	1.9
m4n200Q60A	13.2	0.4	11.8	0.8	12.5	0.4	11.8	0.7
m4n400Q40A	4.7	1.6	3.9	14.0	4.3	3.5	3.9	10.2
m4n400Q60A	14.5	1.2	13.0	4.6	13.7	1.7	13.0	3.7
Avg.	14.8	0.7	13.5	3.7	14.1	1.1	13.5	2.8

greedy procedure and the solution value after the improvement phase.

In addition to the iteration limit MS_0, the phase stops with a time limit of $n/4$ seconds.

4.2. Improvement phase

The improvement phase depends on one parameter M, described in Section 3.2 and intended to allow movements between solutions with higher cost and smaller infeasibility. We made experiments with different values of M and decided to set it to zero. This may be explained because local search procedures rarely find feasible solutions when the initial ones are infeasible, as mentioned before.

To decide the operators to be included in the final version of our algorithm, and the order to apply these operators, we have studied the effect of different combinations in the improvement phase. There are many possible combinations, and we have compared some of them through computational experiments. Table 2 shows the results from four combinations with better results in terms of solution quality and computational time. These combinations include the operators `2-opt`, `fw-ins` and `bw-ins` in this order, and differ in the type of 3-opt moves examined. The table shows, for medium and large instances, the average percentage of deviation from the best known solutions (*avg*) and the computational time (*time*) obtained by the four permutations. Based on these results we decided to include in the final VND procedure only the following operators: `2-opt`, `fw-ins`, `bw-ins` and `3-opt-b`. We consider

that this choice provides a good balance between solution quality and computational time.

This phase is limited in time to at most $n/2$ seconds.

4.3. Perturbation and refinement phase

The time limit for this phase is the total time minus the time consumed by the previous phases.

The parameters of the MIP-based shaking procedure `mip-rand` are set as follows. We impose a time limit of $n/10$ seconds and a limit of 200 branching nodes to the MIP solver on each model. The number of random variables fixed to 0 and fixed to 1 is set to $p_0|A|$ and $p_1 n$ respectively, being $p_0 = 0$ and $p_1 = 0.2$ initially. The values of p_0 and p_1 are increased by 0.05 if the gap between the lower and upper bounds is larger than 20%, they are decreased by 0.05 if the restricted model was optimally solved, and they are unchanged otherwise. This process is repeated until a total time limit is reached or 10 iterations have failed to find a better solution. The random component of the process is aimed to generate different tours, and for that reason `mip-rand` is used as a shaking procedure. We emphasize that the primal-heuristic procedure embedded in the branch-and-cut approach plays an important role in the generation of feasible solutions.

We now describe three different strategies combining the three shaking procedures introduced in Section 3.3.

Strategy I consists of applying `3-opt-rand` and `db-rand` to the current solution. It is an iterative procedure that at iteration l performs $\lfloor (l+3)/2 \rfloor$ moves for $l = 1, \ldots, 10$. A 3-opt move is applied with probability 0.8, thus a double-bridge moves with probability 0.2. We accept a solution from a move even when its cost is larger than the cost of the current solution. However, a worsening is allowed only when the new cost is smaller than 1.05 times the current best value. We must recall also that a solution is also discarded when its infeasibility is larger than the one of the current solution. If a solution from a move is accepted, it replaces the current solution for the next move. The VND in Section 3.2 is applied to the current solution after the last move of the iteration.

Table 3: Computational results to select the strategy for the perturbation phase.

	Strategy I		Strategy II		Strategy III	
	avg	time	avg	time	avg	time
m4n40Q40A	5.2	9.2	5.0	24.6	4.6	18.4
m4n40Q60A	4.0	6.5	0.9	21.8	2.4	17.4
m4n60Q40A	1.0	20.9	10.9	51.5	1.7	47.4
m4n60Q60A	7.1	15.1	4.1	53.7	6.8	41.0
m4n80Q40A	2.7	38.4	12.9	83.7	2.8	81.7
m4n80Q60A	2.7	24.6	8.0	69.8	4.4	57.5
m4n100Q40A	3.4	49.1	16.5	101.6	6.3	101.6
m4n100Q60A	2.9	36.3	7.6	101.0	3.4	89.4
Avg.	3.6	25.0	8.2	63.5	4.0	56.8

Strategy II consists of applying the shaking procedure `mip-rand`. We must recall that it is an iterative procedure that solves mathematical formulations with fixed randomly-selected variables.

Strategy III applies Strategies I and II, one after the other, with half time and iteration limits.

We conducted preliminary computational experiments to compare the three strategies. Table 3 shows average deviations and computing times for small and medium size instances ($n \leq 100$). Each row corresponds to 10 runs of the algorithm over each instance. When n is small and the vehicle capacity is not tight (for example, $Q = 60$), the three strategies produced similar results, and it is difficult to say if one is better than the others. However, on larger instances (say $n \geq 80$) the first strategy outperforms the other two. This is mainly due to the fact that `mip-rand` typically achieves its time limit with solutions of poor quality for large instances.

Based on these preliminary experiments, the next section analyzes the three strategies for small instances and the algorithm with the first strategy for large instances.

5. Computational Results

To evaluate the performance of the algorithm we have considered the set of 240 small instances already used in [16] with $n = 30$, generated using a method similar to that proposed by Mosheiov [18]. A new set of medium and large size instances is generated in the same way.

Table 4: Computational results for small instances.

branch-and-cut		Strategy I			Strategy II			Strategy III		
status	# instances	improv	avg	time	improv	avg	time	improv	avg	time
Optimum	206	1866	0.09	3.6	2009	0.02	5.9	2023	0.02	6.6
No optimum	34	314	-5.56	6.8	319	-5.56	17.8	331	-5.67	17.4

For each number n of locations, the generator produces $n-1$ random pairs of coordinates (points) in the square $[-500, 500] \times [-500, 500]$, and the depot is located in $(0, 0)$. The travel cost c_{ij} is computed as the Euclidean distance between the points i and j. The depot does not demand any commodity (i.e. $q_1^k = 0$ for all $k \in K$). The demands q_i^k are randomly generated in $[-10, 10]$ for all $1 < i < n$ and all $k \in K$. For each $k \in K$, the value q_n^k is defined to ensure $\sum_{i \in V} q_i^k = 0$. If $q_n^k \notin [-10, 10]$ then the random generator is applied to produce new q_i^k values. Observe that $-10m \leq \sum_{k \in K} q_i^k \leq 10m$ for all i. For that reason we have considered instances with $Q \geq 10m$. We generated a group of four random instances (A,B,C,D) for each value $m \in \{3, 4, 5\}$, each $n \in \{100, 200, 400\}$ and each $Q \in \{10m, 10m + 20, 10m + 40\}$. Therefore, our benchmark collection of medium and large m-PDTSP instances contains 108 items.

The branch-and-cut algorithm described in [16] was able to find the optimal solutions for some small instances with $n \in \{20, 25, 30\}$, $m \in \{2, 3\}$, and $Q \in \{10, 12, 15, 17, 20, 25\}$. Table 4 summarizes and compares the computational results given by the branch-and-cut algorithm and the heuristic approaches on the set of instances with $n = 30$. The branch-and-cut algorithm was able to solve to optimality 206 instances out of the 240 items (see the first two columns). From the results for these instances, there is not a clear difference between the three shaking strategies. We show the results for the three implementations. Each implementation was run 10 times over each instance, thus each heuristic approach was run 2060 times on instances where the optimal solution is known, and 340 times on instance without optimality proofs. Columns *improv* displays the number of times that the objective value of an implementation was equal or better than the one of the branch-and-cut algorithm. Similarly, Columns *avg* and *time* give the average deviation and the average time, respectively. It is interesting to observe that the three implementations were successful

in finding better solutions on difficult instances for the branch-and-cut approach. Strategy III (i.e. the one combining the three shaking procedures) is the one that performs slightly better, finding $2023 + 331$ times a solution that improves or equals the one given by the branch-and-cut algorithm.

To see the effect of the vehicle capacity, we conducted some experiments on an instance with $n = 25$, $m = 3$ and different capacities. It is a variant of an instance given by Mosheiov [18] for a related pickup-and-delivery problem. Table 5 shows the computational results of the three implementations compared with the results of the branch-and-cut algorithm. The results are divided in two parts. The top part of the table shows the results obtained when the initial load of the vehicle is unfixed, and the bottom part shows the results when the vehicle leaves the depot with empty load. The time limit of the branch-and-cut algorithm was 10000 seconds. It could not certificate that the solutions of the two instances with Q=9 were optimum solutions. Regarding the heuristic implementations, the difficulty of obtaining a solution increases when the vehicle capacity decreases. This is due to the complexity of checking the feasibility of tours. In addition, requiring an empty initial load reduces the feasibility region, and makes the problem slightly more difficult. Strategy III gives the lowest average deviation of 0.09%, while Strategy II gives the second lowest of 0.16% and Strategy I gives 0.28%. Regarding computational time, Strategy I was faster than the others on these instances.

Table 6 shows the computational results obtained by using our implementation with Strategy I on the set of 108 instances with medium and large size. For these instances the optimal solution is not known. Each row corresponds to 10 runs of the algorithm over an instance. Column headings stand for the best solution value found in the 10 runs (*best*), the gap between the average and best values (*avg*), and the average computational time (*time*) of a run. The table shows that the algorithm is quite robust in the sense that the deviation between the best and average solution values is around 3%, never larger than 7%. Another aspect to point out is that the best solution values for those instances with tight capacity ($Q = 10m$) are usually much larger, almost double, than the best solution values for the instances with loose capacity ($Q = 10m + 40$). This fact indicates that the difficulty of the

Table 5: Computational results for the Mosheiov (1994) instance: $m = 3$ and $n = 25$.

		B&C		Strategy I			Strategy II			Strategy III		
load	Q	opt	time	best	avg	time	best	avg	time	best	avg	time
No	19	4431	0.3	0.00	0.00	2.2	0.00	0.00	0.3	0.00	0.00	1.2
	18	4493	0.9	0.00	0.00	2.3	0.00	0.00	0.4	0.00	0.00	1.4
	17	4572	1.1	0.00	0.00	2.3	0.00	0.00	1.1	0.00	0.00	1.9
	16	4614	1.2	0.00	0.00	2.4	0.00	0.00	1.7	0.00	0.00	2.3
	15	4744	7.4	0.00	0.00	2.4	0.00	0.00	2.0	0.00	0.00	3.2
	14	4744	3.4	0.00	0.00	2.4	0.00	0.00	1.3	0.00	0.00	2.4
	13	4926	10.6	0.00	0.00	2.5	0.00	0.00	4.3	0.00	0.00	4.3
	12	5047	36.8	0.00	0.00	2.8	0.00	0.00	5.8	0.00	0.00	5.2
	11	5458	936.3	0.00	0.00	2.8	0.00	0.00	7.1	0.00	0.00	6.8
	10	5692	598.8	0.00	0.00	3.5	0.00	0.00	5.9	0.00	0.00	7.1
	9	6028	10000.0	0.60	0.60	3.3	0.00	0.24	9.6	0.00	0.18	10.1
	8	6350	4811.3	0.00	0.01	4.0	0.00	0.05	11.6	0.00	0.00	12.0
	7	6843	2906.8	1.36	1.98	4.1	0.00	1.36	14.1	0.00	1.05	11.7
Avg.			1485.8	0.15	0.20	2.8	0.00	0.13	5.0	0.00	0.09	5.4
Yes	21	4915	5.8	0.00	0.00	2.8	0.00	0.00	1.5	0.00	0.00	2.8
	20	5119	7.7	0.00	0.23	3.0	0.00	0.00	3.9	0.00	0.00	4.9
	19	5119	12.5	0.00	0.23	3.0	0.00	0.00	4.5	0.00	0.00	4.9
	18	5119	9.6	0.00	0.00	3.0	0.00	0.00	4.5	0.00	0.00	5.3
	17	5119	12.0	0.00	0.00	3.0	0.00	0.00	4.3	0.00	0.00	4.7
	16	5119	11.6	0.00	0.64	3.0	0.00	0.00	6.0	0.00	0.00	6.8
	15	5173	13.9	0.00	0.95	3.1	0.00	0.39	7.4	0.00	0.00	7.5
	14	5295	31.3	0.00	0.92	3.2	0.00	1.38	6.8	0.00	1.07	8.0
	13	5376	37.7	0.00	0.00	3.2	0.00	0.00	6.8	0.00	0.00	7.5
	12	5639	242.7	0.00	0.09	3.4	0.00	0.00	9.2	0.00	0.00	10.0
	11	5852	1533.7	0.00	0.26	4.1	0.00	0.26	14.0	0.00	0.04	10.8
	10	6113	4914.8	0.00	0.38	4.1	0.00	0.00	9.6	0.00	0.00	13.3
	9	6430	10000.0	0.00	0.00	4.0	0.00	0.15	14.7	0.00	0.00	12.2
	8	6600	6449.7	0.36	0.41	4.6	0.00	0.15	14.8	0.00	0.11	14.4
	7	6911	1590.9	0.00	1.19	5.3	0.00	0.56	14.3	0.00	0.00	15.8
Avg.			1658.3	0.02	0.35	3.5	0.00	0.19	8.1	0.00	0.08	8.6
Avg. over all			1578.2	0.08	0.28	3.2	0.00	0.16	6.7	0.00	0.09	7.1

problem increases notably when the capacity is tighter. Regarding computational times, they get larger when n increases, as expected, but they also augment when the number m of products to transport increases and the vehicle capacity Q is tight. This might be due to the fact that the number of k-opt feasibility checks increases. Finally, there are several entries in the table where the computational time exceeds the time limit of n seconds. This occurs because the stopping criterion on a time limit is performed after the end of a shaking procedure.

Finally, although the algorithm presented in this paper has not been designed to tackle the 1-PDTSP, it can be applied to that problem since it is a particular case of the m-PDTSP. Table 7 compares the performance of our approach and other heuristic algorithms described in the literature to solve the 1-PDTSP. GRASP/VND refers to the algorithm proposed by Hernández-Pérez et al. [12]; GA refers to the Genetic Algorithm proposed by Zhao et al. [26]; Seq-GVND and Mix-GVND refer to the algorithms based on the General Variable Neighborhood Search proposed by Mladenović et al. [17]; Strategy I refers to our implementation where the shaking phase uses the first strategy described in Section 4.3. Each row shows the computational results for ten 1-PDTSP instances with a given number of customers n and capacity Q. We report results taken from [17], where the best and average values for GRASP/VNS and GA are based on 10 runs, and for Seq-GVND and Mix-GVND on 20 runs. In addition the table reports the results of our implementation based on 20 runs. For each approach and each row in the table, *best* is the average gap between the best value generated by that approach on each instance in that row, and the best value generated by all the approaches on the same instance. Similarly, *avg* is the average gap between the average value generated by that approach on each instance in that row, and the best value generated by all the approaches on the same instance. Finally, *time* is the average computational time. For the GA algorithm, there are not results for the instances with $Q > 10$, and therefore the cells in the table are empty. Observe that the our approach (Strategy I) gives better results than the GRAS/VND and GA methods. However, the GVND algorithms given by Mladenović et al. [17] outperform our approach. Nevertheless, recall that ours is designed to tackle a more general problem, and for this reason it does

Table 6: Computational results for medium and large instances.

			$m=3$			$m=4$			$m=5$		
n	Q	seed	best	avg	time	best	avg	time	best	avg	time
100	$10m$	A	17545	1.6	48.8	18636	3.4	49.4	22473	3.1	81.3
		B	16926	2.1	45.7	17861	4.5	48.8	21513	1.4	67.7
		C	16925	2.6	41.1	19692	1.5	57.9	22071	6.9	56.8
		D	17504	4.4	43.0	21045	2.6	58.1	23045	3.6	63.4
	$10m+20$	A	11202	2.1	27.4	12304	2.9	36.4	14002	5.1	41.6
		B	10888	2.3	28.6	11657	3.1	35.4	12963	5.7	42.4
		C	11155	1.5	26.5	12584	2.3	35.6	14165	3.0	43.1
		D	10867	3.4	27.1	13801	0.9	37.9	15281	2.8	45.7
	$10m+40$	A	9026	3.8	24.4	10119	2.5	30.3	12177	2.1	37.2
		B	8919	2.9	24.2	9461	2.9	29.1	10274	2.9	34.6
		C	9238	2.3	21.7	10120	2.2	26.3	11191	2.1	34.3
		D	9158	1.4	23.3	10592	1.9	29.8	11331	6.0	35.4
Avg.				2.5	31.8		2.6	39.6		3.7	48.6
200	$10m$	A	27989	4.5	158.2	35954	2.7	191.7	38468	3.7	200.7
		B	33067	2.5	181.6	37856	1.7	196.7	43917	0.9	214.2
		C	28404	3.4	169.0	34352	1.7	188.6	38833	2.4	205.7
		D	32538	2.4	170.2	40343	3.2	194.8	44402	1.7	219.8
	$10m+20$	A	17672	2.3	82.7	22535	2.8	116.5	24959	4.6	165.3
		B	20217	1.7	90.9	23535	5.6	138.4	28029	2.4	169.1
		C	17783	2.5	89.7	22165	2.1	125.7	25881	2.0	177.3
		D	20409	2.0	95.6	25601	2.3	152.7	28527	3.4	180.2
	$10m+40$	A	14110	2.0	74.0	17828	1.6	90.4	20316	1.7	109.1
		B	15735	3.0	80.2	19366	0.9	102.2	22049	2.4	114.4
		C	14026	2.2	75.5	17120	0.9	98.0	19743	4.0	111.2
		D	15658	2.7	78.1	19250	4.6	98.1	22095	3.2	115.5
Avg.				2.6	112.1		2.5	141.1		2.7	165.2
400	$10m$	A	45914	1.4	407.6	58168	3.0	456.0	68424	2.9	537.4
		B	51346	1.8	415.0	57315	3.2	434.9	64838	2.7	483.1
		C	46346	2.8	403.9	56262	2.6	430.8	63350	4.1	528.8
		D	51357	5.0	421.5	60369	2.0	449.9	65049	5.8	481.1
	$10m+20$	A	28176	3.7	268.3	35374	4.0	388.7	43642	1.6	420.0
		B	31571	1.9	320.1	36008	2.7	405.3	40318	4.7	411.7
		C	28957	1.6	250.8	34290	2.4	383.6	39463	3.1	403.4
		D	31968	2.1	321.2	38258	1.7	396.0	44041	2.1	406.9
	$10m+40$	A	22148	4.7	210.3	28078	2.6	296.3	33120	3.8	395.0
		B	24928	3.0	223.9	28469	2.9	278.9	32794	1.9	379.1
		C	22316	5.1	201.6	27613	1.4	273.1	31376	2.2	365.0
		D	25394	2.1	225.6	30243	2.1	338.7	34734	2.3	395.0
Avg.				2.9	305.8		2.6	377.7		3.1	433.9
Avg. over all				2.7	149.9		2.5	186.1		3.2	215.9

Table 7: Computational results for 1-PDTSP instances.

		GRASP/VND			GA		Seq-GVNS			Mix-GVNS			Strategy I		
n	Q	best	avg	time	best	avg	best	avg	time	best	avg	time	best	avg	time
100	10	1.87	4.66	8.9	0.96	1.83	0.27	1.52	10.0	0.00	0.34	23.8	0.10	0.57	29.0
	20	0.68	2.58	2.2			0.07	1.00	5.9	0.04	0.22	15.0	0.07	0.52	10.6
	40	0.00	0.56	0.7			0.00	0.19	1.8	0.00	0.04	6.0	0.00	0.00	8.4
200	10	4.89	7.60	41.8	2.72	4.12	0.46	2.27	49.7	0.01	1.07	75.7	1.55	2.81	49.3
	20	4.56	6.85	17.4			0.50	2.32	37.6	0.00	0.90	67.6	2.08	3.59	21.7
	40	1.34	3.16	4.3			0.08	0.97	18.0	0.00	0.31	42.3	0.29	1.02	16.1
300	10	5.32	7.71	117.9	4.18	5.58	0.85	2.40	104.6	0.00	1.46	122.8	2.45	3.85	97.7
	20	6.67	8.68	50.9			0.93	2.66	38.7	0.05	1.48	115.9	3.66	5.63	49.5
	40	2.86	4.80	12.9			0.42	1.40	24.9	0.01	0.56	88.6	1.43	2.76	33.8
400	10	5.81	7.85	220.4	4.09	5.89	0.35	2.05	165.9	0.11	1.22	165.4	2.57	4.28	160.8
	20	6.51	8.62	91.7			0.61	2.07	69.4	0.00	1.31	152.4	4.20	6.00	85.8
	40	3.41	5.21	23.9			0.51	1.30	48.7	0.01	0.71	144.3	1.95	3.27	59.7
500	10	5.95	7.93	391.0	5.64	7.45	0.13	1.65	124.1	0.20	1.40	209.8	2.86	4.72	265.3
	20	6.63	8.51	164.8			0.25	1.78	107.0	0.09	1.46	194.8	3.89	5.98	143.8
	40	4.48	6.11	44.0			0.43	1.52	89.8	0.02	0.88	193.5	2.70	4.22	98.9
Avg.		4.07	6.05	79.5	3.52	4.98	0.39	1.67	59.7	0.04	0.89	107.9	1.99	3.28	75.3

not take advantage of certain particularities of the case $m = 1$. For example, the GVND algorithm uses the fact that if the load of the vehicle when leaving a customer is identical for two customers, they can be interchanged in a k-opt move without requiring the feasibility check. This happens frequently when solving 1-PDTSP instances where the demands of the customers have been generated in the interval [-10, 10].

6. Conclusions

This paper proposes a heuristic approach to solve the multi-commodity Pickup-and-Delivery Traveling Salesman Problem. This problem is an interesting and complex routing problem as it generalizes (and is related to) many other variants that have been addressed in the vehicle routing literature during recent years. It concerns the problem of finding a minimum-cost Hamiltonian route for a capacitated vehicle that must collect and deliver several products, each one with possibly several origins and several destinations. We describe and analyze a hybrid approach that merges improvement operators and shaking procedures, one of them based on solving a MIP model. The algorithm has proved to be effective to

solve difficult instances with up to 400 customers and 5 products.

References

[1] S. Anily and J. Bramel. Approximation algorithms for the capacitated traveling salesman problem with pickups and deliveries. *Naval Research Logistics*, 46(6):654–670, 1999.

[2] C. Archetti and M. G. Speranza. Vehicle routing problems with split deliveries. *International Transactions in Operational Research*, 19(1-2):3–22, 2012.

[3] C. Bordenave, M. Gendreau, and G. Laporte. A branch-and-cut algorithm for the nonpreemptive swapping problem. *Naval Research Logistics*, 56(5):478–486, 2009.

[4] P. Chalasani and R. Motwani. Approximating capacitated routing and delivery problems. *SIAM Journal on Computing*, 28(6):2133–2149, 1999.

[5] D. Chemla, F. Meunier, and R. Wolfler-Calvo. Bike hiring system: Solving the static rebalancing problem. *Discrete Optimization*, 10(2):120–146, 2013.

[6] G. Erdoğan, J. F. Cordeau, and G. Laporte. A branch-and-cut algorithm for solving the non-preemptive capacitated swapping problem. *Discrete Applied Mathematics*, 158(15):1599–1614, 2010.

[7] M. Fischetti, J.J. Salazar González, and P. Toth. A branch-and-cut algorithm for the symmetric generalized traveling salesman problem. *Operations Research*, 45(3):378–394, 1997.

[8] G. Ghiani and G. Improta. An efficient transformation of the generalized vehicle routing problem. *European Journal of Operational Research*, 122(1):11–17, 2000.

[9] G. Guastaroba and Speranza M.G. Kernel search: An application to the index tracking problem. *European Journal of Operational Research*, 217(1):54–68, 2012.

[10] P. Hansen and N. Mladenović. Variable neighborhood search: Principles and applications. *European Journal of Operational Research*, 130(3):449–467, 2001.

[11] P. Hansen and N. Mladenović. First vs. best improvement: An empirical study. *Discrete Applied Mathematics*, 154(5):802–817, 2006.

[12] H. Hernández-Pérez, I. Rodríguez-Martín, and J. J. Salazar-González. A hybrid grasp/vnd heuristic for the one-commodity pickup-and-delivery traveling salesman problem. *Computers and Operations Research*, 36(5):1639–1645, 2009.

[13] H. Hernández-Pérez and J. J. Salazar-González. Heuristics for the one-commodity pickup-and-delivery traveling salesman problem. *Transportation Science*, 38(2):245–255, 2004.

[14] H. Hernández-Pérez and J. J. Salazar-González. The one-commodity pickup-and-delivery traveling salesman problem: Inequalities and algorithms. *Networks*, 50(4):258–272, 2007.

[15] H. Hernández-Pérez and J. J. Salazar-González. The multi-commodity one-to-one pickup-and-delivery traveling salesman problem. *European Journal of Operational Research*, 196(3):987–995, 2009.

[16] H. Hernández-Pérez and J. J. Salazar-González. The multi-commodity pickup-and-delivery traveling salesman problem. *Networks*, 63(1):46–59, 2014.

[17] N. Mladenović, D. Urošević, S. Hanafi, and A. Ilić. A general variable neighborhood search for the one-commodity pickup-and-delivery travelling salesman problem. *European Journal of Operational Research*, 220(1):270–285, 2012.

[18] G. Mosheiov. The travelling salesman problem with pick-up and delivery. *European Journal of Operational Research*, 79(2):299–310, 1994.

[19] G. Nagy and S. Salhi. Heuristic algorithms for single and multiple depot vehicle routing problems with pickups and deliveries. *European Journal of Operational Research*, 162(1):126–141, 2005.

[20] M. Nowak, O. Ergun, and C. C. White. An empirical study on the benefit of split loads with the pickup and delivery problem. *European Journal of Operational Research*, 198(3):734–740, 2009.

[21] H. N. Psaraftis. k-interchange procedures for local search in a precedence-constrained routing problem. *European Journal of Operational Research*, 13(4):391–402, 1983.

[22] H. N. Psaraftis. A multi-commodity, capacitated pickup and delivery problem: The single and two-vehicle cases. *European Journal of Operational Research*, 215(3):572–580, 2011.

[23] T. Raviv, M. Tzur, and I. A. Forma. Static repositioning in a bike-sharing system: models and solution approaches. *EURO Journal on Trasnportation and Logistics*, 2(3):187–229, 2013.

[24] I. Rodríguez-Martín and J. J. Salazar-González. Hybrid heuristic approaches for the multi-commodity one-to-one pickup-and-delivery traveling salesman problem. *Journal of Heuristics*, 18(6):849–867, 2011.

[25] J. J. Salazar-González and B. Santos-Hernández. The split-demand one-commodity pickup-and-delivery travelling salesman problem. *Transportation Research Part B*, 75:58–73, 2015.

[26] F. Zhao, S. Li, J. Sun, and D. Mei. Genetic algorithm for the one-commodity pickup-and-delivery traveling salesman problem. *Computers and Industrial Engineering*, 56(4):1642–1648, 2009.