The Traveling Salesman Problem with Pickups, Deliveries, and Draft Limits

Author 1: Enrico Malaguti

Author 2: Silvano Martello

Author 3: Alberto Santini

Journal: Omega (vol. 74), pp. 50–58 (2018).

The Traveling Salesman Problem with Pickups, Deliveries, and Draft Limits

Enrico Malaguti[1], Silvano Martello[2], and Alberto Santini[3]

[1]Università di Bologna*
[2]Università di Bologna†
[3]Università di Bologna‡

December 29, 2022

Abstract

We introduce a new generalization of the traveling salesman problem with pickup and delivery, that stems from applications in maritime logistics, in which each node represents a port and has a known draft limit. Each customer has a demand, characterized by a weight, and pickups and deliveries are performed by a single ship of given weight capacity. The ship is able to visit a port only if the amount of cargo it carries is compatible with the draft limit of the port. We present an integer linear programming formulation and we show how classical valid inequalities from the literature can be adapted to the considered problem. We introduce heuristic procedures and a branch-and-cut exact algorithm. We examine, through extensive computational experiments, the impact of the various cuts and the performance of the proposed algorithms.

1 Introduction

One of the most well known variants of the (asymmetric) *Traveling Salesman Problem* (TSP) is the *TSP with Pickup and Delivery* (TSPPD). The problem is defined on a directed graph $G = (N, A)$ with node set $N = \{0, 1, \ldots, n, n+1, \ldots 2n, 2n+1\}$ and arc set $A = \{(i, j) : i, j \in N\}$. Node 0 is the starting depot and node $2n+1$ is the ending depot (that can eventually coincide). Each arc $(i, j) \in A$ has a cost $c_{ij} \geqslant 0$, and we assume that the *triangle inequality* ($c_{ij} \leqslant c_{ik} + c_{kj}$ $\forall i, j, k \in N$) holds. One has to serve n customers, each of which is associated with a *pickup* node i and a *delivery* node j. We assume, without loss of generality that, for any customer i, the pickup node i is in $\{1, \ldots, n\}$, and the corresponding delivery node j coincides with $n + i$. The objective is to find a Hamiltonian path of minimum total cost that starts at node 0 and terminates at node $2n + 1$, in which the pickup node of every customer is visited before the corresponding delivery node. Although a customer may be origin or destination of a number of different requests, we always associate two distinct nodes to each request.

In the *capacitated* TSPPD (sometimes referred to in the literature as *the* TSPPD),

(i) each customer has a *demand* d_i, defined by a positive value (*weight*) associated with his pickup node i. We conventionally associate $d_{n+i} = -d_i$ with the corresponding delivery node. (For the depot, we assume $d_0 = d_{2n+1} = 0$.);

(ii) pickups and deliveries are performed by a single vehicle of capacity Q;

(iii) at no time during the tour the total load of the vehicle can exceed Q;

(iv) the vehicle leaves and returns to the depot empty.

*enrico.malaguti@unibo.it
†silvano.martello@unibo.it
‡a.santini@unibo.it

In this paper we consider a generalization of the capacitated TSPPD that stems from maritime applications, in which nodes represent ports. Each node $i \in \{1, \ldots, 2n\}$ has a draft limit $l_i > 0$. In maritime terminology the draft is the distance between the waterline and the bottom of the hull of a ship, and it varies as a function of the cargo onboard the ship. If the draft of a ship is greater than the draft limit of a port, the ship is not able to enter and operate safely at that port (see Figure 1). A ship could then deliver part of its cargo at other ports, until its draft is small enough to allow a visit to the port. The relationship between the amount of cargo onboard and the draft of a ship is given, and therefore the draft limit l_i can be expressed with the same unit as the demands d_i and the capacity Q. In other words, for the *Traveling Salesman Problem with Pickups, Deliveries and Draft Limits* (TSPPDD) it must also hold that

(v) when traveling along arc (i, j), the total load of the ship cannot exceed $\min(l_i, l_j)$.

We assume in the following, without loss of generality, that demands, ship capacity, and drafts are positive integers.

The impact of drafts on maritime logistics is becoming more and more important, as the average size of the vessels is increasing. While draft was traditionally an issue related mostly with tankers and bulk vessels, it now involves container ships as well: the average size of a container ship has increased by 19% just in the four years between January 2007 and January 2011 (see Notteboom and Vernimmen [11]). Upgrading port infrastructure is, most of the time, too expensive and time consuming to be considered a feasible solution. Therefore, the burden of ensuring a proper balance between the economy of scale provided by the bigger vessels and the feasibility of the fleet composition and route planning, is left with the ship operator. As observed by Tirschwell [17],

It's a lot easier for a carrier CEO to sign an order for a new ship than for a port to deepen its draft so that ships can enter or leave fully loaded. One takes 10 minutes, the other 10 years.

To the best of our knowledge, this is the first study on the TSPPDD, although the constraints we impose have been separately considered by other authors.

Dumitrescu, Ropke, Cordeau, and Laporte [6] studied the polytope of the TSPPD, derived facet-defining

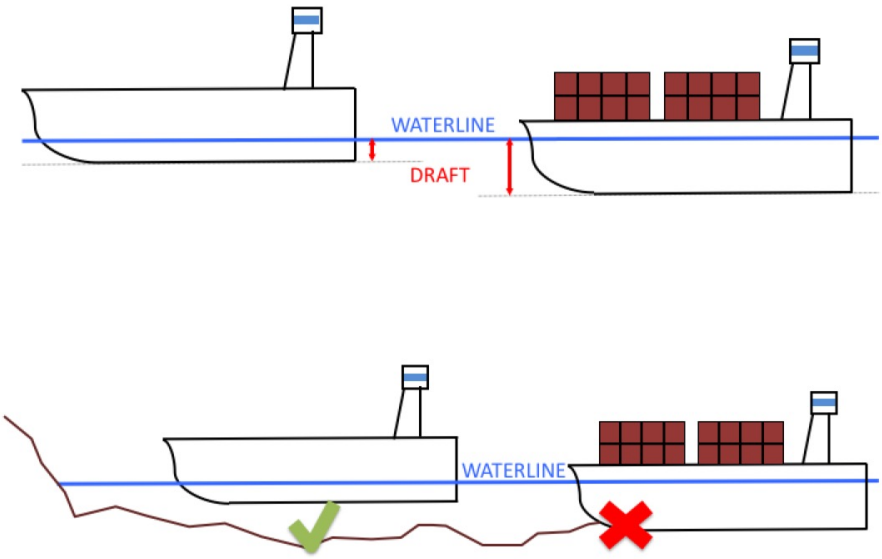

Figure 1: In the picture above, the draft of a ship as a function of the cargo on board. In the picture below, a ship able to enter a port (left) and one whose draft is too large to enter the same port (right).

inequalities, and developed a branch-and-cut algorithm in which the inequalities are separated heuristically. They solved to optimality instances with up to 35 origin-destination pairs.

Ropke, Cordeau, and Laporte [15] and Ropke and Cordeau [14] studied the pickup and delivery problem with time windows, i.e., a multi-vehicle generalization of the TSPPD in which customers can only be visited within their opening time. The former paper presents a branch-and-cut algorithm, while the latter improves on it, by using a branch-and-cut-and-price approach. The traveling salesman problem with draft limits was introduced by Glomvik Rakke, Christiansen, Fagerholt, and Laporte [7]. In this problem, the ship starts from the depot completely loaded and the objective is to find the shortest Hamiltonian path to satisfy the demands of the customers without violating the drafts limits. They proposed two formulations, a branch-and-cut algorithm, and a method to strengthen the bounds through the solution of knapsack problems. The approach was tested on 240 instances with up to 48 nodes, derived from the TSP Library.

Battarra, Pessoa, Subramanian, and Uchoa [3] investigated the same problem, proposing mathematical formulations as well as a branch-and-cut and a branch-and-cut-and-price algorithm. The latter algorithm proved to be very effective and solved to optimality all the instances proposed in [7].

A constraint that can remind our draft constraint has been considered by Ma, Cheang, Lim, Zhang, and Zhu [10], who studied a vehicle routing problem with link capacity constraints, in which road links (i.e., arcs) have limitations on the tonnage of the vehicles allowed to travel along them.

Differently from other generalizations of the TSP (see, e.g., Cordeau, Nossack, and Pesch [5]), the TSPPDD does not have a natural decomposition into simpler problems. In the next section we present a mathematical model for the TSPPDD. In Section 3 we obtain a number of valid inequalities that are used in Section 4 to obtain a branch-and-cut algorithm. In order to provide a good initial solution to the algorithm, a heuristic and a local search approach are proposed in Section 5. Computational experiments are presented in Section 6, and conclusions follow in Section 7.

2 Mathematical model

In this section we present an *Integer Linear Programming* (ILP) formulation of the TSPPDD, and we show how it can be simplified through arc removal.

2.1 Integer Linear Program

For each arc $(i, j) \in A$, let x_{ij} be a binary variable taking the value 1 if and only if arc (i, j) is part of the solution, and y_{ij} be an integer variable representing the quantity of cargo on board the ship when traveling along arc (i, j).

Let us define two parameters, λ_{ij} and υ_{ij}, to represent a lower and an upper bound, respectively, on y_{ij}. The former can be defined as

$$\lambda_{ij} = \begin{cases} d_i & \text{if } i \in \{1, \ldots, n\} \text{ and } j \in \{1, \ldots, n\} \cup \{n + i\} \; ; & (1) \\ -d_j & \text{if } i, j \in \{n + 1, \ldots, 2n\}; & (2) \\ d_i - d_j & \text{if } i \in \{1, \ldots, n\} \text{ and } j \in \{n + 1, \ldots, 2n\} \backslash \{n + i\}; & (3) \\ 0 & \text{otherwise.} & (4) \end{cases}$$

In case (1) i is an origin and j is either another origin or the destination of i: a ship traveling along (i, j) must carry at least the cargo picked up at i. In case (2) both i and j are destinations: the cargo destined to j must be on board when traveling along (i, j). In case (3) i is an origin and j is a destination either than that of i: a ship traveling along (i, j) must carry both the cargo picked up at i and the one to be delivered at j. Finally, if i is a destination and j is an origin, the ship could possibly be empty.

An obvious upper bound on y_{ij} is $\min\{l_i, l_j, Q\}$. A tighter bound may be obtained by decreasing these three quantities as

$$\upsilon_{ij} = \min\{l_i + \min\{0, d_i\}, l_j - \max\{0, d_j\}, Q - \max\{0, -d_i, d_j\}\} \quad (5)$$

Indeed: (i) if i is a destination then the minimum between l_i and Q may be decreased by the amount of cargo delivered at i; (ii) if j is an origin then the minimum between l_j and Q may be decreased by the amount of cargo to be picked up at j.

The TSPPDD can then be formally defined through the following *Integer Linear Programming* (ILP) model:

$$\min \sum_{i \in N} \sum_{j \in N} c_{ij} x_{ij} \tag{6}$$

$$\text{s.t.} \sum_{j \in N} x_{ij} = 1 \qquad (i = 0, \ldots, 2n) \tag{7}$$

$$\sum_{i \in N} x_{ij} = 1 \qquad (j = 1, \ldots, 2n+1) \tag{8}$$

$$\lambda_{ij} x_{ij} \leqslant y_{ij} \leqslant \upsilon_{ij} x_{ij} \qquad (i, j = 1, \ldots, 2n) \tag{9}$$

$$\sum_{j \in N} y_{ij} - \sum_{j \in N} y_{ji} = d_i \qquad (i = 1, \ldots, 2n) \tag{10}$$

$$\sum_{j \in N} y_{0j} = 0 \tag{11}$$

$$\sum_{j \in S} x_{ij} \geqslant 1 \qquad (i = 1, \ldots, n; S \subset N : i \notin S \text{ and } n + i \in S) \tag{12}$$

$$\sum_{j \in S} x_{ij} \geqslant 1 \qquad (i = n+1, \ldots, 2n; S \subset N : i \notin S \text{ and } 2n + 1 \in S) \tag{13}$$

$$x_{ij} \in \{0, 1\}, y_{ij} \in \mathbb{N} \qquad (i, j = 0, \ldots, 2n+1). \tag{14}$$

The objective function (6) minimizes the total cost of the route. Constraints (7) and (8) ensure that the ship starts from depot 0 and ends at depot $2n + 1$ after having visited every port exactly once. Constraints (9) guarantee the feasibility of the quantity of cargo onboard at any time. Constraints (10) impose that all pickups and deliveries be fulfilled. Constraint (11) ensures that the ship starts its route with no load. The precedence constraints (12) enforce each origin to be visited before the corresponding destination, while constraints (13) impose that depot $2n + 1$ be visited after all destinations. Note that constraints (7), (8), (12) and (13) together ensure that the classical subtour elimination constraints be satisfied.

2.2 Arc removal due to precedence, capacity and draft constraints

The ILP model can be enhanced by removing arcs from set A according to the following considerations:

- self-loop arcs (i, i) $(i \in N)$ are not considered;
- arcs of the form $(0, n + i)$ or $(i, 2n + 1)$ $(i \in \{1, \ldots, n\})$ cannot be part of a feasible solution, as they would violate precedence constraints;
- arcs of the form $(n + i, i)$ $(i \in \{1, \ldots, n\})$ would make no sense in a solution;
- arcs of the form (i, j) $(i, j \in \{1, \ldots, n\})$ such that $d_i + d_j > \min\{l_j, Q\}$ would violate either the draft limit at j or the ship capacity;
- arcs of the form $(n + i, n + j)$ $(i, j \in \{1, \ldots, n\})$ such that $d_i + d_j > \min\{l_{n+i}, Q\}$ would violate either the draft limit at $n + i$ or the ship capacity;
- arcs of the form $(i, n + j)$ $(i, j \in \{1, \ldots, n\}, j \neq i)$ such that $d_i + d_j > \min\{l_i, l_{n+j}, Q\}$ would violate either the draft of i, or the draft of $n + j$, or the ship capacity.

3 Valid inequalities

The TSPPDD is as a generalization of the TSPPD which, in turn, is a special case of the *Precedence Constrained TSP* (PCTSP) in which the solution must satisfy precedence relations $i \prec j$ imposed to a set of node pairs. A number of valid TSPPD or PCTSP inequalities are either valid or can be adapted to the TSPPDD, as well as to other related problems (see, e.g., Xue, Luo, and Lim [18]). We considered in particular subtour-elimination, generalized order, capacity and fork cuts.

3.1 Subtour elimination cuts

Given a set $S \subset N$, let $A(S) = \{(i,j) : i,j \in S\}$ and $\bar{S} = N \backslash S$. The classical TSP facet-defining subtour-elimination cut is

$$\sum_{(i,j)\in A(S)} x_{ij} \leqslant |S| - 1 \quad \forall S \subset N. \tag{15}$$

We will adopt the notation of Cordeau [4], namely:

$$\sigma(S) = \{i \in N \ : \ n+1 \leqslant i \leqslant 2n \ \text{ and } \ i - n \in S\} \ \ (\textit{successor nodes});$$
$$\pi(S) = \{i \in N \ : \ 1 \leqslant i \leqslant n \ \text{ and } \ n + i \in S\} \qquad (\textit{predecessor nodes}).$$

Balas, Fischetti, and Pulleyblank [2] have lifted (15) for the PCTSP through the precedence constraints. As each node (but the depots) is the predecessor or successor of exactly one other node, (15) can be lifted in two ways. Let $\delta(S,T) = \{(i,j) \in A : i \in S, j \in T\}$. For predecessors, we have:

$$\sum_{(i,j)\in A(S)} x_{ij} + \sum_{(i,j)\in\delta(S\cap\pi(S),\bar{S}\backslash\pi(S))} x_{ij} + \sum_{(i,j)\in\delta(S,\bar{S}\cap\pi(S))} x_{ij} \leqslant |S| - 1 \quad \forall S \subset N, \tag{16}$$

while for successors we have

$$\sum_{(i,j)\in A(S)} x_{ij} + \sum_{(i,j)\in\delta(\bar{S}\backslash\sigma(S),S\cap\sigma(S))} x_{ij} + \sum_{(i,j)\in\delta(\bar{S}\cap\sigma(S),S)} x_{ij} \leqslant |S| - 1 \quad \forall S \subset N. \tag{17}$$

Consider the relaxation of the TSPPDD obtained by eliminating the constraints on draft limits and ship capacity. The resulting problem is a special case of the PCTSP, and hence inequalities (16) and (17) are valid for the TSPPDD as well.

Another TSP facet-defining cut can be found by a different lifting of (15). Given a set $S \subset N$ with $h = |S| \geqslant 3$, and any ordering of its nodes $S = \{i_1, \ldots, i_h\}$, Grötschel and Padberg [8] proved that the following inequalities are valid for the TSP:

$$\sum_{k=1}^{h-1} x_{i_k,i_{k+1}} + x_{i_h,i_1} + 2\sum_{k=2}^{h-1} x_{i_k,i_1} + \sum_{k=3}^{h-1}\sum_{l=2}^{k-1} x_{i_k,i_l} \leqslant |S| - 1. \tag{18}$$

$$\sum_{k=1}^{h-1} x_{i_k,i_{k+1}} + x_{i_h,i_1} + 2\sum_{k=3}^{h} x_{i_1,i_k} + \sum_{k=4}^{h}\sum_{l=3}^{k-1} x_{i_k,i_l} \leqslant |S| - 1. \tag{19}$$

The dial-a-ride problem is a routing problem in which one has to design vehicle routes and schedules for a set of requests which specify pickup and delivery between origins and destinations. Cordeau [4] proved that, for such problem, the above cuts can be further strengthened by adding a term that takes into account the resulting precedence constraints, obtaining:

$$\sum_{k=1}^{h-1} x_{i_k,i_{k+1}} + x_{i_h,i_1} + 2\sum_{k=2}^{h-1} x_{i_k,i_1} + \sum_{k=3}^{h-1}\sum_{l=2}^{k-1} x_{i_k,i_l} + \sum_{j\in\bar{S}\cap\sigma(S)} x_{j,i_1} \leqslant |S| - 1, \tag{20}$$

$$\sum_{k=1}^{h-1} x_{i_k,i_{k+1}} + x_{i_h,i_1} + 2\sum_{k=3}^{h} x_{i_1,i_k} + \sum_{k=4}^{h}\sum_{l=3}^{k-1} x_{i_k,i_l} + \sum_{j\in\bar{S}\cap\pi(S)} x_{i_1,j} \leqslant |S| - 1. \tag{21}$$

Since the precedence constraints of the dial-a-ride problem are the same as those of the TSPPDD, these cuts are also valid for our problem.

3.2 Generalized order cuts

Another family of valid inequalities, called *generalized m-order constraints*, was introduced by Ruland and Rodin [16] for the TSPPD. Given m disjoint subsets $S_1, \ldots, S_m \subset N$ such that none of them contains 0 or $2n+1$, if it is possible to find a sequence of nodes $i_1, \ldots, i_m \in \{1, \ldots, n\}$ such that:

$$\begin{aligned} i_k &\in S_k \quad (k = 1, \ldots, m), \\ n + i_{k+1} &\in S_k \quad (k = 1, \ldots, m-1), \\ n + i_1 &\in S_m, \end{aligned}$$

then the following inequality is valid:

$$\sum_{l=1}^{m} \sum_{(i,j)\in A(S_l)} x_{ij} \leqslant \sum_{l=1}^{m} |S_l| - m - 1. \tag{22}$$

It has been proved in [4] that, by taking into account the precedences induced by pickup and delivery, these cuts can be lifted in two ways:

$$\sum_{l=1}^{m} \sum_{(i,j)\in A(S_l)} x_{ij} + \sum_{l=2}^{m-1} x_{i_1,i_l} + \sum_{l=3}^{m} x_{i_1,n+i_l} \leqslant \sum_{l=1}^{m} |S_l| - m - 1; \tag{23}$$

$$\sum_{l=1}^{m} \sum_{(i,j)\in A(S_l)} x_{ij} + \sum_{l=2}^{m-2} x_{n+i_1,i_l} + \sum_{l=2}^{m-1} x_{n+i_1,n+i_l} \leqslant \sum_{l=1}^{m} |S_l| - m - 1. \tag{24}$$

Again, the validity for the TSPPDD comes from the consideration that the precedence constraints of the two problems coincide.

3.3 Capacity-draft cuts

Given a subset $S \subset N$, let $d(S) = \sum_{i\in S} d_i$. Consider a set S such that $d(S) > 0$, and define the *reduced capacity* with respect to S as $Q(S) = \min(Q, \max_{i\in S}\{l_i\})$ (upper bound on the load when visiting a node of S). An immediate lower bound on the number of times a vehicle must visit S is then

$$\sum_{(i,j)\in\delta(S,\bar{S})} x_{ij} = \sum_{(i,j)\in\delta(\bar{S},S)} x_{ij} \geqslant \lceil d(S)/Q(S) \rceil. \tag{25}$$

Following Ropke, Cordeau, and Laporte [15], cut (25) can be strengthened by considering two sets $S, T \subset N$ with $q(S) > 0$, and defining $U = \pi(T)\backslash(S \cup T)$. We obtain

$$\sum_{(i,j)\in A(S)} x_{ij} + \sum_{(i,j)\in A(T)} x_{ij} + \sum_{(i,j)\in\delta(S,T)} x_{ij} \leqslant |S| + |T| - \left\lceil \frac{d(S) + d(U)}{Q(S \cup T)} \right\rceil, \tag{26}$$

which coincides with the cut obtained by [15], with the only difference that $Q(S \cup T)$ replaces Q.

3.4 Fork cuts

Consider any routing problem in which a feasible path $P = (k_1, \ldots, k_r)$ becomes infeasible if two nodes $i \in S$ and $j \in T$ ($S, T \subset N$), are added at the beginning and at the end of P. Then the *fork inequality*

$$\sum_{i\in S} x_{i,k_1} + \sum_{h=1}^{r-1} x_{k_h,k_{h+1}} + \sum_{j\in T} x_{k,j} \leqslant r \tag{27}$$

obviously holds. It has been shown in [15] that (27) can be strengthened through sets of nodes that produce intermediate infeasible paths. Specifically we consider subsets S, $T_1, \ldots, T_r \subset N$ such that

$k_h \notin T_{h-1}$ for $h = 2, \dots r$. If the path $(i, k_1, \dots, k_h, j)$ is infeasible for any $h \leqslant r$ and any pair ($i \in S$, $j \in T_h$), then the *outfork inequality*

$$\sum_{i \in S} x_{i,k_1} + \sum_{h=1}^{r-1} x_{k_h,k_{h+1}} + \sum_{h=1}^{r} \sum_{j \in T_h} x_{k_h,j} \leqslant r \tag{28}$$

prohibits infeasible paths obtained by prematurely leaving P. Exactly in the same way one can derive *infork* inequalities by prohibiting infeasible paths obtained by entering P at an intermediate node. As these cuts are valid for any routing problem in which one can decide whether a certain path is infeasible, they hold for the TSPPDD as well.

4 Branch-and-cut algorithm

We implemented a branch-and-cut algorithm based on the root-node formulation (6)-(14). At the root node we relax constraints (12)-(13), which impose precedence and subtour-elimination. At each decision node, we separate those inequalities that are violated by the current (fractional) solution. In addition to these two families of constraints, which ensure feasibility, we generate the cuts described in Section 3. The branch-decision tree exploration is managed by a general purpose software (e.g, CPLEX). In this section we describe how the model was strengthened and how the cuts were separated.

4.1 Strengthened model

In order to strengthen the root-node formulation, we added two sets of constraints to the relaxed model.

Classical *2-cycle elimination* constraints

$$x_{ij} + x_{ji} \leqslant 1 \quad \forall\ (i,j) \in A : j > i \ \text{ and } \ (j,i) \in A. \tag{29}$$

Property 4.1. *In spite of their simplicity, constraints* (29) *are not implied by the relaxed model* (6)-(11), (14). *Indeed*

Proof. It is enough to consider the case $i \leqslant n$, $j > n$, $j \neq n + i$, $d_j = -d_i$. Solution $x_{ij} = x_{ji} = 1$, $y_{ij} = d_i$, $y_{ji} = 0$ does not violate (10), but it violates (29). □

There are $O(n^2)$ potential 2-cycle elimination constraints, hence their addition to the model is not computationally heavy. The experiments showed however that they have limited impact on the solution quality, so we developed the following specialized constraints, that gave much better results.

2) Draft oriented *2-path elimination* constraints

$$x_{ij} + x_{jk} \leqslant 1 \quad \forall\ i,j,k \in \{1, \dots, 2n\} : \text{certain conditions (see below) hold.} \tag{30}$$

Property 4.2. *Inequalities* (30) *are valid for the following cases (corresponding to the enumeration of all possible characterizations of i, j, k), in which a path (i, j, k) would violate either a draft (cases 1-6) or a precedence (cases 7 and 8) constraint (see Figure 2, where pickup nodes are drawn bigger than delivery nodes, and the value on an arc gives the minimum load the ship would have when traveling along it):*

1. $i \leqslant n, j \leqslant n, k \leqslant n$ *and* $d_i + d_j + d_k > \min(Q, l_k)$.
2. $i \leqslant n, j \leqslant n, k > n, k \neq n + i, k \neq n + j$ *and either* $d_i + d_j - d_k > \min(Q, l_j, l_k)$ *or* $d_i - d_k > \min(Q, l_i, l_j)$;
3. $i \leqslant n, j > n, k \leqslant n, j \neq n + i$ *and* $d_i + d_k > \min(Q, l_k)$;
4. $i \leqslant n, j > n, k > n, j \neq n + i, k \neq n + i$ *and either* $d_i - d_j - d_k > \min(Q, l_i, l_j)$ *or* $d_i - d_k > \min(Q, l_j, l_k)$;

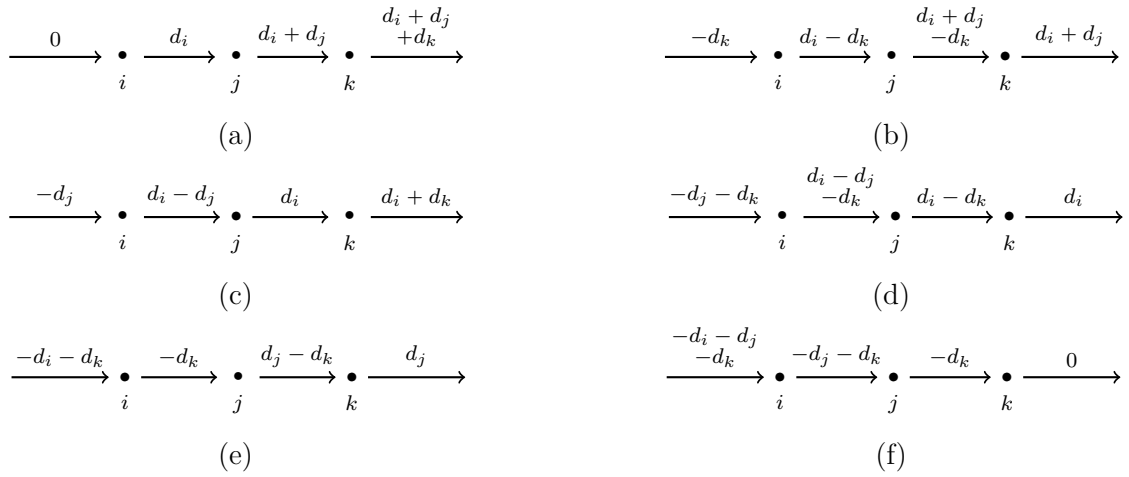

Figure 2: Minimum load on board a ship traveling along arcs (i, j) and (j, k).

5. $i > n, j \leqslant n, k > n, k \neq n + j$ *and* $-d_i - d_k > \min(Q, l_i)$;
6. $i > n, j > n, k > n$ *and* $-d_i - d_j - d_k > \min(Q, l_i)$;
7. $i > n, j \leqslant n, k \leqslant n$ *and* $i = n + k$;
8. $i > n, j > n, k \leqslant n$ *and* $i = n + k$;

Proof. Consider Case 1: the load on the arc leaving k would be at least $d_i + d_j + d_k$ (Figure 2a). Very similar reasonings, immediately emerging from the figures, prove: Case 2 (Figure 2b), and note that the last condition is equivalent to $d_i - d_k > l_i$); Case 3 (Figure 2c); Case 4 (Figure 2d, and note that the last condition is equivalent to as $d_i - d_k > l_k$); Case 5 (Figure 2e); Case 6 (Figure 2f). In cases 7 and 8 no draft violation occurs, but the precedence condition between i and k would be violated. □

The number of potential 2-path constraints is $O(n^3)$ but their inclusion into the model proved to be effective. Note in addition that, as these constraints represent incompatibilities between pairs of arcs, it would be possible to aggregate some of them into stronger clique inequalities, representing incompatibilities between subsets of arcs. This is however automatically done by the solver we used (CPLEX), so there would be no advantage in doing it explicitly.

4.2 Cut separation

The *precedence inequalities* (12) and (13) can both be separated exactly in polynomial time through series of max-flow problems. Violated inequalities (12) can be found by solving n max-flow problems from i to $n + i$ $(i = 1, \ldots, n)$, where the arc capacities are the values of variables x_{ij}. Violated inequalities (13) can be found by solving, in an analogous way, n max-flow problems from $n+i$ to $2n+1$ $(i = 1, \ldots, n)$. Details on these separation methods can be found, e.g., in Padberg and Hong [12].

All the cuts discussed in Section 3 were instead separated in a heuristic way. A heuristic separation method for *subtour elimination cuts* (16) and (17) was given by [4]. Observe that, for any set $S \neq \varnothing$, the arcs incident with all nodes of S can either belong to $\delta^+(S)$, or to $\delta^-(S)$, or to $A(S)$ (in which case they appear twice), and hence

$$\sum_{(i,j)\in\delta^+(S)\cup\delta^-(S)} x_{ij} + 2 \sum_{(i,j)\in A(S)} x_{ij} = 2\,|S|. \tag{31}$$

By combining (31) with twice (16) in one case, and twice (17) in the other, one obtains

$$\sum_{(i,j)\in\delta^+(S)\cup\delta^-(S)} x_{ij} - 2 \sum_{(i,j)\in\delta(S\cap\pi(S),\bar{S}\setminus\pi(S))} x_{ij} - 2 \sum_{(i,j)\in\delta(S,\bar{S}\cap\pi(S))} x_{ij} \geqslant 2 \tag{32}$$

$$\sum_{(i,j)\in\delta^+(S)\cup\delta^-(S)} x_{ij} - 2 \sum_{(i,j)\in\delta(\bar{S}\setminus\sigma(S),S\cap\sigma(S))} x_{ij} - 2 \sum_{(i,j)\in\delta(\bar{S}\cap\sigma(S),S)} x_{ij} \geqslant 2 \tag{33}$$

We therefore heuristically search for subsets S violating (32) or (33), using the simple Tabu search scheme proposed by Augerat [1] for the capacitated vehicle routing problem. Consider the separation of (16) through (32). The search starts from an empty set S and iteratively adds or removes elements from S, trying to minimize the left hand side of (32). When a node is removed from S, its insertion is marked as tabu for a certain number of iterations. In addition, at each iteration, if $|S| \geqslant 3$, the current set S is also used to check whether (20) is violated: in fact, we can choose i_1 of (20) as the node with the largest outflow and compute the left-hand side of (20) by numbering all other nodes at random. A similar procedure is used for separating (17) through (33) as well as, if $|S| \geqslant 3$, for checking whether (21) is violated.

We separate *generalized order cuts* (23) and (24) only for $m = 3$ and $|S_l| = 2$ ($l = 1, 2, 3$) as, for larger values, they become computationally very expensive. Notice that in this case sets S_l can be written as:

$$S_1 = \{i_1, n + i_2\}, \quad S_2 = \{i_2, n + i_3\}, \quad S_3 = \{i_3, n + i_1\}$$

and equation (23) becomes:

$$x_{i_1,n+i_2} + x_{n+i_2,i_1} + x_{i_2,n+i_3} + x_{n+i_3,i_2} + x_{i_3,n+i_1} + x_{n+i_1,i_3} + x_{i_1,i_2} + x_{i_1,n+i_3} \leqslant 2. \tag{34}$$

For every possible choice of $i_1 \in \{1, \ldots, n\}$, we find the node $i_2 \in \{1, \ldots, n\}$ such that the three terms containing only indices $i_1, i_2, n + i_2$ in the lhs of (34) are maximized. Then, we find the node $i_3 \in \{1, \ldots, n\}$ that maximizes the other five terms. In other words, for (23)

$$i_2 = \arg \max_{1 \leqslant j \leqslant n} \{x_{i_1,n+j} + x_{n+j,i_1} + x_{i_1,j}\}; \tag{35}$$

$$i_3 = \arg \max_{1 \leqslant j \leqslant n} \{x_{i_2,n+j} + x_{n+j,i_2} + x_{j,n+i_1} + x_{n+i_1,j} + x_{i_1,n+j}\}, \tag{36}$$

and analogously, for (24):

$$i_2 = \arg \max_{1 \leqslant j \leqslant n} \{x_{i_1,n+j} + x_{n+j,i_1} + x_{n+i_1,n+j}\}; \tag{37}$$

$$i_3 = \arg \max_{1 \leqslant j \leqslant n} \{x_{i_2,n+j} + x_{n+j,i_2} + x_{j,n+i_1} + x_{n+i_1,j}\}. \tag{38}$$

We separate *capacity-draft cuts* (26) using the procedure detailed in [15] which starts with sets $S = \{i\}$ and $T = \{n + j\}$ for all possible $i, j \in \{1, \ldots, n\}$ and tries to augment these sets at each iteration.

Finally, *fork cuts* are separated in both their basic version (27), and in the strenghtened infork and outfork versions (see (28)). The path $P = (k_1, \ldots, k_r)$ that forms the backbone for the cut is constructed as follows. We fix a node $k_0 \in \{1, \ldots, 2n\}$ and we consider all paths $(k_0, k_1, \ldots, k_r)$ for $r \geqslant 2$, that can be constructed by adding arcs corresponding to base columns of the linear relaxation of the problem. In other words, arc (i, j) is used to extend the path only if $x_{ij} > 0$. For each such path, set T is constructed as

$$T = \{j : j \notin P \text{ and } (k_0, k_1, \ldots, k_r, j) \text{ is infeasible}\},$$

and the corresponding set S is

$$S = \{i : i \notin P \text{ and } (i, k_1, \ldots, k_r, j) \text{ is infeasible for all } j \in T\}.$$

Notice that, by construction, $k_0 \in S$. An inequality (27) is added whenever it is violated by the current choice of P, S, and T. For non-violated inequalities, we attempt lifting into outfork and infork inequalities. For example, we attempt to find a violated outfork inequality (28) by adding, in a greedy way, as many nodes as possible to sets $T_1, \ldots, T_r$. Attempting this procedure for all r values would clearly be computationally too expensive, and hence, on the basis of preliminary experiments, we only considered paths with $r \leqslant 6$. In addition, whenever we check a sub-path for feasibility, we store the result in a hash table from which it can be retrieved at a later time. The feasibility check ensures that no precedence constraint is violated and that the draft limits are respected, by assuming that the ship is has the minimum possible load when it enters the sub-path.

5 Heuristic algorithms

In this section we present the heuristics used to obtain feasible initial solutions to the TSPPDD. We will call an origin-destination pair $(i, n+i)$ a *request*. We will call an *insertion* of a request in a partial path a couple $(p_{\text{orig}}, p_{\text{dest}})$ that indicates the positions in the partial path where, respectively, the origin and the destination of the request are inserted. Our approach consists of two constructive heuristics, followed by a refinement procedure.

5.1 Constructive heuristics

Our constructive heuristics start with an empty path and proceed by inserting one request at a time, until no requests are left (and hence an initial feasible solution has been obtained). We considered two approaches, denoted as *Sorted Insert* and *Best Insert*. In the former approach, the requests are preliminarily ordered according to some score that only depends on the requests themselves, and then are inserted one by one in such order: the current request is inserted in a position chosen according to a heuristic criterion. In the latter approach, at each iteration, each non-inserted request is assigned a score and a possible insertion, and the request with the highest score is correspondingly inserted.

The heuristics build a solution by using two kinds of scores, one related to the requests, and one related to their insertion. The *request scores* are

R1 the cost $c_{i,n+i}$ of the origin-destination arc;

R2 the value $\min(l_i, l_{n+i}) - d_i$ of the additional load the ship can carry when entering the two ports.

In order to introduce the insertion scores, let us define, for a path P:

- $c_P = \sum_{(i,j)\in P} c_{ij}$, the cost of the path;
- $d_P = \sum_{(i,j)\in P: 1\leqslant i\leqslant n} d_i$, the total load picked up along the path;
- $w_P = \sum_{(i,j)\in P}(\min\{Q, l_i, l_j\} - y_{ij})$, where y_{ij} is the load of the ship when traveling along arc (i, j): w_P represents the waste of capacity along the path.

The *insertion score* is assigned to a possible insertion $(p_{\text{orig}}, p_{\text{dest}})$ by considering the extended path P given by the insertion. Four scores (the lower, the better) were evaluated:

I1 c_P, the cost of the new path;

I2 $c_P d_P$, a measure that favors paths with low cost, while giving priority to requests with low demand;

I3 $c_P + \rho d_P$, where $\rho > 0$ is a prefixed parameter, a measure similar to the previous measure, but with lesser impact of d_P. (We adopted, on the basis of preliminary computational experiments, the value $\rho = 1$);

I4 $c_P w_P$, a measure that favors paths with low cost and high capacity utilization.

Four *Sorted Insert* procedures were obtained by sorting the requests according to decreasing or increasing request score R1 or R2. For each of them, the insertion was decided using, as insertion score, either I1 or I4 (note that I2 and I3 need not be considered, since once the current request has been fixed, d_P is constant for all insertions). In total this results in eight different implementations.

Four *Best Insert* procedures were obtained by respectively evaluating, for each non-inserted request, insertion scores I1-I4. For each of them, two implementations were obtained by selecting the next request and position either as the one providing the smallest insertion score, or the one providing the largest *regret*, i.e., the largest difference between the second minimum and the minimum insertion score (or the insertion score, when only one insertion is feasible). In this case too we thus obtained eight different implementations.

For the values of n we used in our computational experiments, the CPU time taken by these procedures is negligible, hence all of them were executed (and refined, as shown in the next section). Other scores were attempted too, but the sixteen implementations we described were the only non dominated ones.

5.2 Refinement

The feasible solutions produced by the constructive heuristics were improved through a very simple Tabu search, defined by the following ingredients:

- *move*: three-opt (see Lin [9]) with check on the feasibility of the resulting solution. Notice that, for an oriented graph, every triplet of arcs has just one possible recombination;
- *Tabu list*: for each move, the cheapest removed arc is stored;
- *Tabu tenure*: a prefixed parameter (having value 30 in our implementation);
- *halting criteria*: a prefixed maximum number of iterations, or of iterations with no improvement. (We used values 50000 and 500, respectively, in our experiments).

6 Computational experiments

The exact and heuristic approaches of the previous sections were implemented in C++ and run on an Intel Xeon 3.10 GHz with 8 GB RAM, equipped with four cores. In order to allow future fair comparisons, all the experiments were performed by setting to one the number of threads.

We used IBM ILOG CPLEX 12.6 as ILP solver for the branch-and-cut algorithm of Section 4. Remind that we relax the precedence and subtour-elimination inequalities (12)-(13): at each decision node, the inequalities that are violated by the current solution are separated and added via a CPLEX callback. The additional valid inequalities of Section 3 were not generated at each decision node: the decision about separation is taken according to different probabilistic distributions, depending on the number of explored nodes and on the specific cut. Namely, the probability of separation linearly decreases from 1 to α for nodes 1–100, from β to γ for nodes 101–20 000, while it is set to γ for all subsequent nodes. Good values of α, β and γ were determined, through preliminary computational experiments, as

- subtour elimination cuts: $\alpha = 0.9$, $\beta = 0.5$, $\gamma = 0.05$;
- generalized order cuts: $\alpha = 1$ (always separated), $\beta = 1$, $\gamma = 0.1$;
- capacity-draft cuts: $\alpha = 0.75$, $\beta = 0.125$, $\gamma = 0.0125$;
- fork cuts: $\alpha = 0.75$, $\beta = 0.0625$, $\gamma = 0.00625$.

We randomly generated our benchmark starting from the eight instances of the `TSPLIB` [13] that have been used in [7] and in [3] to generate benchmarks for the TSP with draft limits: bayg29, burma14, fri26, gr17, gr21, gr48, ulysses16, and ulysses22. From each TSP instance we obtained TSPPDD instances having $2n + 2$ nodes, with $n \in \{10, 14, 18, 22\}$, as follows. For each value of n,

- a TSP node was randomly selected as the starting and ending depot (TSPPDD nodes 0 and $2n + 1$). Then n origin-destination pairs were randomly selected from the remaining TSP nodes, together with the corresponding costs. A TSP node was allowed to be selected more than once, but not for the same pair;
- the n demands d_j were randomly generated in the interval $[1, 100]$;
- four sets of instances were obtained by setting the ship capacity to $Q = 50\,n\,C$, with $C \in \left\{\frac{1}{10}, \frac{3}{10}, \frac{1}{2}, 2\right\}$, as follows:
  - for each $C \in \left\{\frac{1}{10}, \frac{3}{10}, \frac{1}{2}\right\}$, four instances were produced by: (i) randomly selecting, with probability $\mathcal{P} \in \left\{0, \frac{1}{3}, \frac{2}{3}, 1\right\}$, nodes j ($1 \leqslant j \leqslant 2n$) that will have a binding draft; (ii) randomly generating the draft l_j of each selected node in the interval $[|d_j|, Q-1]$; (iii) setting the draft of the non-selected nodes to Q. Note that, for $\mathcal{P} = 0$, no node has a binding draft, so we can evaluate our methods also on the special case given by a capacitated TSPPD;
  - for the same reason, for $C = 2$, we only generated a single instance with all nodes having draft $Q = 100\,n$, i.e., we obtained an uncapacitated TSPPD instance.

$\|N\|$	C	Basic model		2-cycle		2-path		Subt. elim.		Gen. order		Cap.-draft		Fork		B&C	
		Root	Final	Root	Final	Root	Final	Root	Final	Root	Final	Root	Final	Root	Final	Root	Final
22	0.1	4.06	0.00	3.88	0.00	4.02	0.00	4.01	0.00	4.01	0.00	3.78	0.00	2.65	0.00	2.47	0.00
22	0.3	16.84	2.30	16.69	2.33	18.03	2.22	16.27	1.87	16.42	1.78	16.31	1.76	14.28	0.33	14.20	0.32
22	0.5	20.35	2.98	20.69	3.21	21.06	2.84	19.96	2.10	20.12	3.05	20.09	2.95	19.67	1.92	19.23	1.52
22	2.0	9.99	0.00	10.05	0.00	9.71	0.00	8.74	0.00	9.90	0.00	9.53	0.00	9.77	0.00	8.72	0.00
30	0.1	15.95	6.51	15.56	6.46	15.78	6.63	15.83	6.19	15.87	6.07	15.22	5.65	12.37	2.67	12.35	2.41
30	0.3	27.47	19.35	27.42	19.09	27.45	19.25	27.00	18.23	27.43	18.95	26.97	18.63	26.52	17.11	25.93	16.02
30	0.5	24.34	15.96	24.29	16.20	24.34	16.16	23.71	14.54	24.30	16.03	24.30	16.01	23.96	15.51	23.44	14.01
30	2.0	10.27	0.83	10.28	0.77	10.27	0.74	9.89	0.14	10.20	0.79	10.28	0.82	10.27	0.83	9.89	0.09
38	0.1	19.74	15.07	19.47	15.06	19.74	15.02	19.68	14.88	19.69	14.82	18.92	13.79	16.96	9.45	16.00	8.88
38	0.3	28.74	24.74	28.51	24.74	28.61	24.66	28.55	24.49	28.54	24.78	28.45	24.58	28.20	23.93	27.78	22.39
38	0.5	23.53	19.53	23.63	19.50	23.53	19.54	23.43	19.03	23.15	19.01	23.53	19.37	23.32	18.96	23.15	18.45
38	2.0	10.47	4.16	10.48	4.20	10.47	4.15	10.43	3.71	10.44	3.98	10.43	4.49	10.47	4.62	10.43	4.56
46	0.1	24.67	21.59	24.57	21.51	24.67	21.56	24.61	21.50	24.58	21.61	23.71	20.54	21.14	15.85	19.94	15.17
46	0.3	36.79	34.64	36.74	34.59	36.74	34.49	36.70	34.40	36.79	34.67	35.93	33.54	36.40	34.14	35.25	31.35
46	0.5	29.68	27.40	29.40	27.40	29.56	27.42	29.62	27.07	29.71	27.42	29.57	27.23	29.59	27.37	29.23	26.12
46	2.0	15.27	12.82	15.32	12.79	15.24	12.43	15.23	11.94	15.26	12.66	15.27	12.96	15.27	12.80	15.23	11.94
Average		21.82	14.96	21.72	14.96	21.62	14.69	21.57	14.48	21.70	14.81	21.40	14.51	20.50	13.22	18.33	10.83

Table 1: Effect of elimination constraints and cuts on the percentage gaps between upper and lower bound.

C	$\mathcal{P}$	$\lvert N\rvert = 22$				$\lvert N\rvert = 30$				$\lvert N\rvert = 38$				$\lvert N\rvert = 46$			
		CH	TS	B&C	OPT	CH	TS	B&C	OPT	CH	TS	B&C	OPT	CH	TS	B&C	OPT
0.1	0	0.14	0.00	0.00	8	1.79	1.03	1.03	7	8.86	6.20	6.20	5	12.34	8.05	7.97	0
0.1	0.33	0.37	0.00	0.00	8	1.53	0.16	0.16	7	11.38	8.75	8.75	2	16.39	12.49	12.49	0
0.1	0.67	0.38	0.00	0.00	8	4.44	3.39	3.39	6	13.26	9.71	9.71	3	23.96	19.29	19.29	0
0.1	1	0.10	0.00	0.00	8	5.94	5.07	5.07	4	14.01	10.85	10.85	1	26.00	20.96	20.94	0
0.3	0	0.29	0.00	0.00	8	18.18	16.26	16.19	1	28.04	22.99	22.99	0	37.58	32.36	32.36	0
0.3	0.33	0.91	0.00	0.00	8	20.46	17.19	17.19	0	31.81	26.92	26.92	0	41.52	35.48	35.48	0
0.3	0.67	0.72	0.00	0.00	8	19.38	16.37	16.37	0	29.10	22.50	22.50	0	40.44	31.37	31.37	0
0.3	1	1.74	1.27	1.27	7	18.58	14.35	14.35	0	26.29	17.16	17.16	0	34.27	26.17	26.17	0
0.5	0	4.37	2.14	2.14	5	23.98	21.53	21.53	0	31.98	25.77	25.77	0	39.18	32.94	32.94	0
0.5	0.33	4.37	2.67	2.67	6	20.66	17.60	17.60	0	27.08	24.59	24.59	0	35.84	32.74	32.74	0
0.5	0.67	2.81	0.74	0.73	7	16.32	13.76	13.76	1	19.18	16.61	16.61	0	33.14	23.88	23.88	0
0.5	1	1.80	0.68	0.53	7	4.39	3.14	3.14	4	8.95	6.81	6.81	2	19.47	14.90	14.90	0
2.0	0	0.00	0.00	0.00	8	0.28	0.09	0.09	7	5.58	4.56	4.56	3	13.64	11.94	11.94	1
	Average	1.38	0.58	0.56	7.38	11.99	10.00	9.99	2.85	19.66	15.65	15.65	1.23	28.75	23.27	23.27	0.08
	CPU secs	0.01	2.23	431		0.02	10.32	2398		0.06	32.56	3175		0.14	94.01	3549	

Table 2: Percentage gaps of the upper bounds produced by the constructive heuristic, the Tabu refinement, and the branch-and-cut algorithm with respect to the best lower bound.

In total, we obtained 13 instances for each value of n, i.e., 52 TSPPDD instances for each TSP instance, and hence an overall benchmark of 416 instances. The computer code and the instances are available at `https://github.com/alberto-santini/tsppddl`. The results of the computational experiments are reported in Tables 1 and 2.

Table 1 examines the impact of strengthening constraints (Section 4.1) and valid inequalities (Section 3). The table considers the separate inclusion of each constraint or cut and reports, for each of them, the percentage gaps (at the root node and final, i.e. after 1 hour CPU time) with respect to the best known upper bound. For different values of n and C, the first two columns give the percentage gaps for the basic model (6)-(14), the last two columns give the percentage gaps for the branch-and-cut algorithm (Sections 4 and 5) while the other pairs of columns refer to the separate addition of constraints and cuts. An additional row gives the average gaps over the 416 instances.

The results after 1 hour CPU time (columns 'Final') show that fork cuts are the most powerful inequalities for smaller capacity values, while subtour elimination cuts frequently obtain better results for larger capacities. In a single case ($|N| = 46$, $C = 0.3$) capacity-draft cuts prevail: disaggregated results show that they produce the best gap for 14 instances out of 32. In many cases subtour elimination, generalized order, and capacity-draft cuts produce similar gaps. The results at the root node (columns 'Root') exhibit a similar behavior. The last two columns show that an effective combination of the various cuts within the branch-and-cut algorithm produce by far the best results. There is a single exception for $|N| = 38$ and $C = 2.0$, where subtour elimination beats branch and cut: it must be noted, however, that, as previously described, such capacity value produces uncapacitated TSPPD instances.

Table 2 provides the percentage gaps of the upper bounds with respect to the best lower bound. For different values of C and $\mathcal{P}$, the table contains four groups of four columns (one group for each number of nodes). In each group, the first three columns provide the percentage gaps between the upper bounds produced by the constructive heuristic of Section 5.1 (column CH), the tabu refinement of Section 5.2 (column TS), and the branch-and-cut algorithm (column B&C) with respect to the final lower bound value obtained by the branch-and-cut algorithm of Section 4. The fourth column of each group gives the number of instances (out of 8) solved to proven optimality by the branch-and-cut algorithm. Two additional rows give the average values over the 104 instances generated for each number of nodes, and the average CPU times (in seconds) required by the three algorithms.

The results show that the branch-and-cut algorithm is very effective for the instances with 22 nodes (92% of instances solved), while, as it could be expected, its behavior worsens for larger instances with 30, 38, and 46 nodes (36%, 15%, and 0.01% of instances solved, respectively). The same consideration holds for the B&C optimality gaps. The heuristic algorithms exhibit a satisfactory behavior: within very short CPU times (below 2 minutes, on average), the constructive heuristic and its simple Tabu search refinement give feasible solutions not much worse than those produced by the branch-and-cut algorithm (starting from such solutions) after one hour. By restricting the analysis to the 150 instances for which a provably optimal solution has been obtained, one can observe that the optimality gap of the constructive heuristic was 0.987% and that of the Tabu search refinement was 0.013%. Note however that the CPU time requested by branch-and-cut is not uselessly spent, as it allows to certify optimality or to evaluate the actual optimality gap.

Overall, the outcome of our computational experiments proves that taking into account realistic constraints like ship capacities and draft limits considerably increases the difficulty of finding optimal TSP solutions. Consider for example the line of Table 2 corresponding to $C = 2.0$, i.e., to uncapacitated TSP instances with pickup and delivery, and observe that both the gaps and the numbers of optimally solved instances are considerably better than the average values in the subsequent line. This is also confirmed by the fact that the algorithms in [6] for the TSPPD, as well as those in [3] for the TSPDL were able to solve larger instances of the respective problems. On the other hand, the good performance of the constructive heuristic and of its Tabu search refinement indicate that such algorithms can be profitably used for practical purposes.

7 Conclusions

We have studied for the first time a realistic variant of the classical traveling salesman problem with pickups and deliveries, that arises in maritime transportation. Considering the ship capacities and the draft limits of the ports to be visited is a crucial addition for realistically modeling problems in which one has to schedule the sequence of ports to be visited by a container ship. We have defined an integer linear programming model and we have shown how valid inequalities developed for the traveling salesman and the vehicle routing problem can be adapted to our problem. We have developed heuristic approaches and an exact branch-and-cut algorithm. Extensive computational experiments on instances of realistic size have shown that exactly solving this problem variant is extremely challenging. However, we have seen that approximate solutions of good quality (and hence particularly useful to practitioners) can be obtained within short computing times. Future developments could extend the study to the multi-vehicle case. Indeed, while the tramp shipping business is usually interested in scheduling one ship at a time, liner shipping operators are faced with the problem of planning the routes of a whole fleet.

Acknowledgements

Research supported by Air Force Office of Scientific Research (Grants FA9550-17-1-0025 and FA9550-17-1-0067) and by MIUR-Italy (Grant PRIN 2015).

References

[1] P. Augerat, J.M. Belenguer, E. Benavent, A. Corberán, and D. Naddef. Separating capacity constraints in the CVRP using tabu search. *European Journal of Operational Research*, 106(2-3):546–557, 1998.

[2] E. Balas, M. Fischetti, and W.R. Pulleyblank. The precedence-constrained asymmetric traveling salesman polytope. *Mathematical Programming*, 68(1-3):241–265, 1995.

[3] M. Battarra, A.A. Pessoa, A. Subramanian, and E. Uchoa. Exact algorithms for the traveling salesman problem with draft limits. *European Journal of Operational Research*, 235(1):115–128, 2014.

[4] J-F Cordeau. A branch-and-cut algorithm for the dial-a-ride problem. *Operations Research*, 54(3):573–586, 2006.

[5] M. Cordeau, J. Nossack, and E. Pesch. Mathematical formulations for a 1-full-truckload pickup-and-delivery problem. *European Journal of Operational Research*, 242:1008–1016, 2015.

[6] I. Dumitrescu, S. Ropke, J.-F. Cordeau, and G. Laporte. The traveling salesman problem with pickup and delivery: polyhedral results and a branch-and-cut algorithm. *Mathematical Programming*, 121(2):269–305, 2010.

[7] J. Glomvik Rakke, M. Christiansen, K. Fagerholt, and G. Laporte. The traveling salesman problem with draft limits. *Computers & Operations Research*, 39(9):2161–2167, 2012.

[8] M. Grötschel and M.W. Padberg. Lineare charakterisierungen von travelling salesman problemen. *Zeitschrift für Operations Research*, 21(1):33–64, 1977.

[9] S. Lin. Computer solutions of the traveling salesman problem. *Bell System Technical Journal*, 44(10):2245–2269, 1965.

[10] H. Ma, B. Cheang, A. Lim, L. Zhang, and Y. Zhu. An investigation into the vehicle routing problem with time windows and link capacity constraints. *Omega*, 40(3):336–347, 2012.

[11] T.E Notteboom and B. Vernimmen. The effect of high fuel costs on liner service configuration in container shipping. *Journal of Transport Geography*, 17(5):325–337, 2009.

[12] M. Padberg and S. Hong. On the symmetric travelling salesman problem: A computational study. *Mathematical Programming Study*, 12:78–107, 1980.

[13] G. Reinelt. Tsplib–a traveling salesman problem library. *ORSA Journal on Computing*, 3(4):376–384, 1991.

[14] S. Ropke and J.-F. Cordeau. Branch and cut and price for the pickup and delivery problem with time windows. *Transportation Science*, 43(3):267–286, 2009.

[15] S. Ropke, J.-F. Cordeau, and G. Laporte. Models and branch-and-cut algorithms for pickup and delivery problems with time windows. *Networks*, 49(4):258–272, 2007.

[16] K.S. Ruland and E.Y. Rodin. The pickup and delivery problem: Faces and branch-and-cut algorithm. *Computers & Mathematics with Applications*, 33(12):1–13, 1997.

[17] P. Tirschwell. Berth productivity: The trends, outlook and market forces impacting ship turnaround times. Port Productivity (White paper), pages 1–24. Journal of Commerce, July 2014.

[18] L. Xue, Z. Luo, and A. Lim. Exact approaches for the pickup and delivery problem with loading cost. *Omega*, 59:131–145, 2016.