OJMO

Open Journal of Mathematical Optimization

Nicola Bianchessi, Dario Ostuni & Giovanni Righini

Exact optimization algorithms for an order picking problem

Volume 7 (2026), article no. 3 (18 pages)

https://doi.org/10.5802/ojmo.51

Article submitted on November 26, 2024, revised on July 23, 2025, accepted on May 24, 2026.

Open Journal of Mathematical Optimization is a member of the Centre Mersenne for Open Scientific Publishing
http://www.centre-mersenne.org/
e-ISSN: 2777-5860

Exact optimization algorithms for an order picking problem

Nicola Bianchessi
Department of Computer Science, University of Milan, Via Celoria 18, 20133 Milan, Italy
nicola.bianchessi@unimi.it

Dario Ostuni
Department of Computer Science, University of Milan, Via Celoria 18, 20133 Milan, Italy
dario.ostuni@gmail.com

Giovanni Righini
Department of Computer Science, University of Milan, Via Celoria 18, 20133 Milan, Italy
giovanni.righini@unimi.it

Abstract

We consider a combinatorial optimization problem arising when a set of pick-up and delivery orders must be satisfied within an Automated Storage/Retrieval System. The computational complexity of the problem is still open, but it is conjectured to be NP-hard. We point out some of its relevant properties and we describe three exact optimization algorithms to solve it, one based on dynamic programming and the other two on branch-and-bound. We also present a mixed-integer linear programming model to solve the problem by general purpose mathematical programming solvers. Computational results are provided to assess the effectiveness of these methods.

Digital Object Identifier 10.5802/ojmo.51

2020 Mathematics Subject Classification 90C27.

Keywords Order picking, Integer linear programming, Dynamic programming, Branch-and-bound.

Acknowledgments The authors are grateful to two anonymous reviewers for their constructive and helpful comments. The work was supported by PRIN Project 2022YWWETX by the Italian Ministry for University and Research, 2022. Code, input data and results are publicly available at `https://github.com/dariost/2_1_PD_F`.

1 Introduction

We consider a combinatorial optimization problem arising in the context of Automated Storage and Retrieval Systems (AS/RS) optimization. An AS/RS is a system where boxes containing objects are stored. In our setting the locations are equally spaced along a line, all boxes are identical, they take one location each, and their position along the line is given. At an endpoint of the line an input/output device (called *origin* in the remainder) is located. A shuttle moves along the line; it is equipped with enough space to accommodate two boxes.

Two types of operations must be done: deliveries and pick-ups. A delivery consists of carrying a box from the origin to its location; a pick-up consists of carrying a box from its location to the origin. Both types of operations can be grouped in a same trip of the shuttle, provided that the capacity constraint is satisfied. While pick-ups can be executed in any order, deliveries must comply with the arrival sequence of the boxes to be stored, which is given. This means that incoming boxes are loaded on the shuttle according to a fixed sequence, although it is not mandatory to deliver them immediately. Without loss of generality, it can be assumed that boxes arrive according to their numbering.

The objective is to minimize the total distance travelled by the shuttle.

In spite of being formulated with a highly simplified setting, compared to real AS/RS systems, the problem is conjectured to be NP-hard and its analysis is meant as a first step to address more complex and realistic order picking optimization problems.

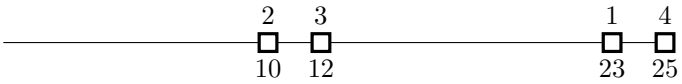

Figure 1 Squares indicate deliveries, whose numbering and distance from the origin are reported respectively above and below the squares. A solution of minimum distance is made by three trips: $\{2\}$, $\{3\}$ and $\{1, 4\}$ with total cost $10 + 12 + 25 = 47$. A solution exists with two trips, $\{1, 2\}$ and $\{3, 4\}$, with total cost equal to $23 + 25 = 48$.

Literature review

For an overview of AS/RS systems and their optimization we refer the reader to surveys, such as [3, 4, 5, 6, 7, 8].

The specific problem studied in this paper was considered by Barbato et al. [1] and classified as $2/1/PD/F$, meaning that the shuttle has capacity 2, the arrangement of locations is mono-dimensional (along a line), both pickups and deliveries are involved and the position where each operation must be done is fixed.

In Barbato et al. [1] several optimal order picking problems were considered and their computational complexity was established, with two notable exceptions: one is problem $q/1/P/V$ (where V stands for variable positions of the operations), that was later proven to be polynomially solvable by Barbato et al. [2]; the other is $2/1/PD/F$, whose complexity is still open.

In this paper we present three exact optimization algorithms to solve $2/1/PD/F$ effectively, although not in guaranteed polynomial time. We also formulate the problem as a MILP to solve it by available MILP solvers.

Paper outline

Section 2 establishes some relevant properties of the problem $2/1/PD/F$, that are exploited to design the algorithms. Section 3 recalls a polynomial-time dynamic programming algorithm that solves problem $2/1/D/F$ (i.e. with no pickups) to proven optimality and illustrates a similar dynamic programming algorithm that solves $2/1/PD/F$ to proven optimality, with complexity that is polynomial in the number of deliveries but exponential in the number of pickups. Sections 4 and 5 illustrate two branch-and-bound algorithms. Finally, in Section 6 we present a mixed-integer linear programming model, that can be solved by general-purpose MILP solvers. Computational results are reported in Section 7.

2 Problem properties

Two indexed sets $\mathcal{P}$ and $\mathcal{D}$ of operations are given, representing pickups and deliveries, respectively. The distance of each location from the origin is indicated by π_i for $i \in \mathcal{P}$ and δ_i for $i \in \mathcal{D}$. In the remainder, by "closer" and "farther", we mean closer or farther to the origin. By *trip* we mean the subset of pickup and delivery operations executed by the shuttle between two consecutive visits to the origin.

▶ **Observation 1.** *The cost of each trip of the shuttle is given by twice the maximum distance between the origin and the locations visited either for pickup or delivery.*

For the sake of simplicity, from now on we get rid of the factor 2 and we consider the maximum distance to be reached as the trip cost.

Although delivery boxes must be loaded on the shuttle according to their given arrival order, the availability of two units of capacity on the shuttle allows for some flexibility in the order in which boxes are delivered. In particular, it is allowed receiving a delivery box $j \in \mathcal{D}$ without immediately delivering it, then receiving delivery boxes $j+1, j+2, \ldots, j+k-1$ and delivering them one by one in $k-1$ distinct trips and finally delivering boxes j and $j+k$ in a single trip. This can be optimal when δ_j and δ_{j+k} are relatively large, while $\delta_{j+1}, \ldots, \delta_{j+k-1}$ are relatively small.

A direct consequence is the following.

▶ **Observation 2.** *A minimum total distance solution does not necessarily minimize the number of trips.*

An example is shown in Figure 1.

In case a delivery box j loaded on the shuttle is not immediately delivered in the next trip, then the number of allowed operations in the trip is reduced to one pickup and one delivery, instead of two, because one unit of capacity is taken by box j for the whole duration of the trip and therefore it is unavailable. In this case we say that the *trip capacity* is equal to 1 or the trip is *single-capacity*; otherwise, a trip is said to be *double-capacity*.

▶ **Observation 3.** *Four types of trips are possible:*

- *A-trips of capacity* 2*, with two delivery boxes on board initially, visiting two delivery locations;*
- *B-trips of capacity* 2*, with one delivery box on board initially, visiting one delivery location;*
- *C-trips of capacity* 1*, with two delivery boxes on board initially, visiting one delivery location;*
- *D-trips of capacity* 2*, with no delivery boxes on board initially, visiting no delivery location.*

A-trips, B-trips and D-trips can be assigned up to two pickups, while C-trips can be assigned at most one pickup. Figure 2 shows a small instance whose optimal solution is made by four trips of four different types.

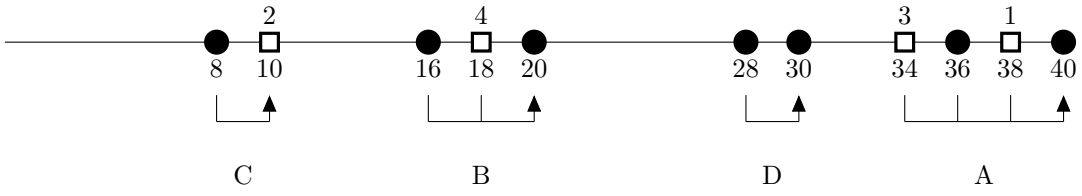

Figure 2 A small example: the optimal solution is made by four trips of four different types. Dots indicate pickups, squares indicate deliveries, whose numbering is reported above the squares. Below the line, optimal trips are represented: for each trip, an arrow indicates the farthest location to be visited.

▶ **Observation 4.** *The feasibility of a trip does only depend on the number of operations assigned to it.*

In any feasible trip of type A, B, C or D, as defined above, the operations can be feasibly sequenced so that all deliveries are done first, sorted by non-decreasing distance, and all pickups follow, sorted by non-increasing distance. Owing to this observation, we can concentrate on the subset of pickups and deliveries assigned to each trip, disregarding their sequence.

▶ **Property 1.** *Every C-trip must visit a delivery location between two delivery locations assigned to a same trip of capacity* 2*, because this is the only case in which the trip capacity is reduced to* 1*. By converse, every time a trip of capacity* 2 *visits two non-consecutive deliveries j' and j'' with $j'' > j' + 1$, all deliveries k such that $j' < k < j''$ must be assigned to C-trips.*

An example is given by the A-trip and the C-trip represented in Figure 2.

▶ **Property 2.** *All pairs of deliveries, say (i', i'') and (j', j''), assigned to different trips with capacity* 2 *must be disjoint, i.e.* $\max\{i', i''\} < \min\{j', j''\}$ *or viceversa.*

These two properties are trivially implied by the constraint on the shuttle capacity.

▶ **Property 3.** *There exists an optimal solution in which at most one of any two consecutive deliveries is assigned to a B-trip.*

Proof. Any solution with two B-trips each one visiting one of two consecutive deliveries can be transformed into a non-worse solution, by re-assigning the closest delivery $j \in \mathcal{D}$ from its trip t to the trip t' visiting the other one. The cost of trip t cannot increase, since it looses a visit; the cost of trip t' does not increase either, because it already visits another delivery that is farther than j. ◀

An example is shown in Figure 3.

Let c_t^D the *delivery cost* of a trip $t \in T$, i.e. the maximum distance of the delivery locations visited by t.

▶ **Property 4.** *There exists an optimal solution in which each B-trip $t = t_B$ visiting a delivery $j \in \mathcal{D}$ and for each A-trip $t' = t_A$ visiting delivery $j - 1$ or $j + 1$, $c_{t'}^D \geq \delta_j$.*

Proof. Any solution violating this property, so that $c_{t'}^D < \delta_j$, can be transformed into a non-worse solution complying with the property, by re-assigning delivery $j + 1$ or $j - 1$ to the B-trip. Let k be the re-assigned delivery. By deleting k from t', the cost of t' cannot increase. When k is inserted in t (that becomes an A-trip), the cost of t does not increase either, because (i) $\delta_k \leq c_{t'}^D$, since t' visits k, (ii) $c_{t'}^D < \delta_j$ for the assumption and (iii) $\delta_j = c_t^D$ because t is a B-trip. Hence $\delta_k < c_t^D$. ◀

An illustration is given in Figure 4.

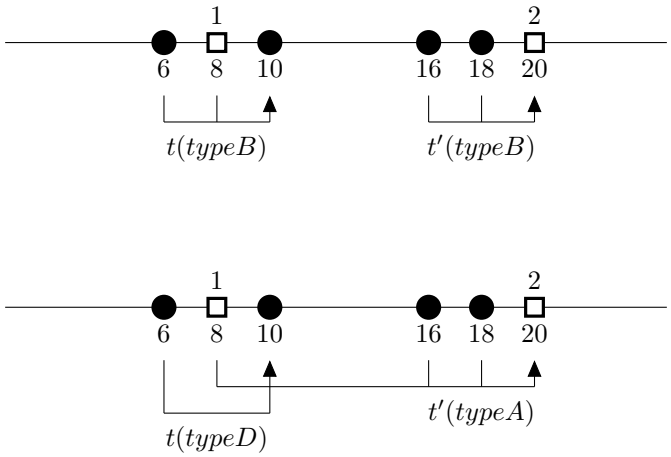

Figure 3 Two B-trips visiting two consecutive delivery locations (top) can be replaced without worsening the cost (bottom).

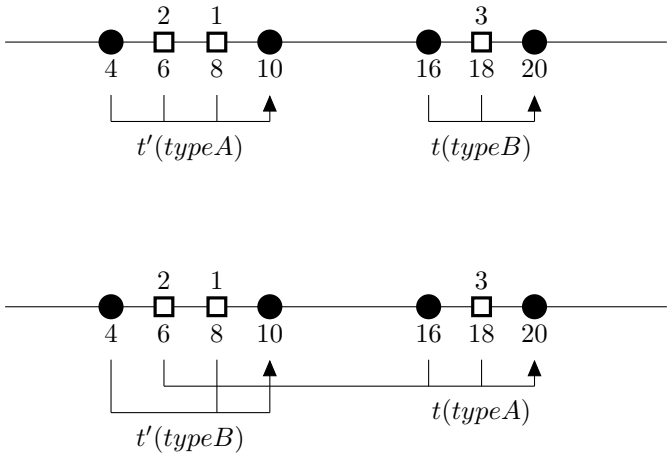

Figure 4 Top: a solution violating Property 4. Bottom: A non-worse solution obtained from it and complying with Property 4.

▶ **Property 5.** *There exists an optimal solution in which for any C-trip visiting delivery $k \in \mathcal{D}$ and for any A-trip visiting deliveries $j' \in \mathcal{D}$ and $j'' \in \mathcal{D}$ with $j' < k < j''$, $\delta_k < \min\{\delta_{j'}, \delta_{j''}\}$.*

Proof. Any solution violating this property can be transformed into a non-worse solution by swapping $k \in D$ and the closest (to the origin) among j' and j''. ◀

An illustration is given in Figure 5.

Now we consider the sub-problem of optimally assigning pickups to trips. Consider a given assignment of deliveries to trips. Let $\mathcal{T}$ be the set of T trips defined by such an assignment. Renumber all trips by non-decreasing delivery cost c^D such that $c^D_1 \leq c^D_2 \leq \cdots \leq c^D_T$. Some of them have capacity 1 (C-trips) and some have capacity 2 (A- and B-trips); we indicate their capacity by $cap(t)$ for $t \in \mathcal{T}$. Consider the set $\mathcal{P}$ of P pickups and renumber them by non-decreasing distance π such that $\pi_1 \leq \pi_2 \leq \cdots \leq \pi_P$.

▶ **Property 6.** *If $\pi_P \leq c^D_T$, then it is optimal to assign the $cap(T)$ farthest pickup orders to trip T.*

Proof. Any solution violating this property can be transformed into a non-worse solution, by swapping the $cap(T)$ farthest pickups with those assigned to trip T, if any. The cost of the trips different from T involved in the exchange cannot increase, since farther pickups are replaced by closer pickups or no pickups; the cost of trip T does not increase either, because it is given by c^D_T independently of the pickups assigned to it. ◀

An illustration is given in Figure 6.

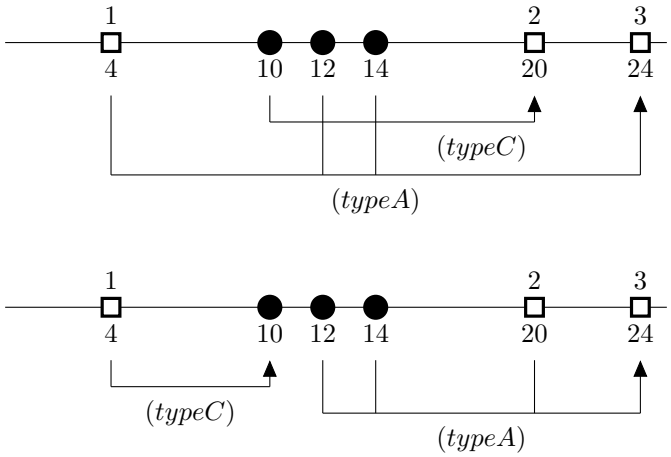

Figure 5 Top: a solution violating Property 5. Bottom: A non-worse solution complying with Property 5.

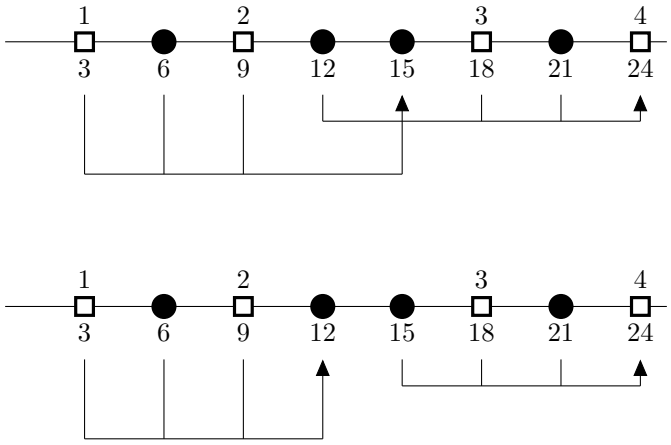

Figure 6 Top: a solution violating Property 6. Bottom: A non-worse solution complying with Property 6.

▶ **Property 7.** *If $\pi_P > c_T^D$ and $cap(T) = 2$, then it is optimal to assign the two farthest pickups (if there are two) to trip T.*

Proof. Any solution violating this property can be transformed into a non-worse solution, by swapping the two farthest pickups with those assigned to trip T, if any. (In case $P = 1$, the following proof still holds, by assuming the existence of a dummy pickup at distance 0.) Let p be the farthest pickup assigned to trip T, if any. Let t be the trip that pickup P is assigned to. When the pickups assigned to t and T are swapped, the cost of trip t decreases by $\pi_P - \max\{\pi_p, c_t^D\}$, while the cost of trip T increases by $\pi_P - \max\{\pi_p, c_T^D\}$. Therefore the total cost of the solution decreases by $\max\{\pi_p, c_T^D\} - \max\{\pi_p, c_t^D\}$ which is non-negative, because $c_T^D \geq c_t^D$.

An illustration is given in Figure 7. ◀

▶ **Property 8.** *If $\pi_{P-1} \leq c_T^D$ and $cap(T) = 1$, then it is optimal to assign the farthest pickup to trip T.*

Proof. Any solution violating this property can be transformed into a non-worse solution, by swapping the farthest pickup with the one assigned to trip T, if any. Let $p \leq P - 1$ be the pickup assigned to trip T, if any. Let t be the trip that pickup order P is assigned to. Let p' the other pickup assigned to t, if any. Consider two cases.

Case (i): $\pi_P \leq c_T^D$**.** When pickups p and P are swapped between trips t and T, the cost of trip t decreases by $\max\{c_t^D, \pi_P\} - \max\{c_t^D, \pi_p, \pi_{p'}\}$, while the cost of trip T remains equal to c_T^D. Therefore the total cost of the solution decreases because $\pi_P \geq \max\{\pi_p, \pi_{p'}\}$ (for the ordering of pickups).

Case (ii): $\pi_P \geq c_T^D$**.** When pickups p and P are swapped between trips t and T, the cost of trip t decreases by $\pi_P - \max\{c_t^D, \pi_p, \pi_{p'}\}$, while the cost of trip T increases by $\pi_P - c_T^D$. Therefore the total cost of the solution

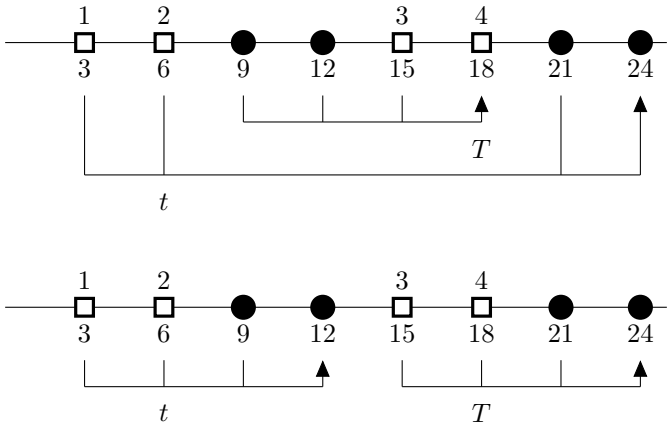

Figure 7 Top: a solution violating Property 7. Bottom: A non-worse solution complying with Property 7.

decreases by $c_T^D - \max\{c_t^D, \pi_p, \pi_{p'}\}$ which is non-negative because $c_T^D \geq c_t^D$ (for the ordering of T, since $t < T$) and $\pi_p \leq c_T^D \quad \forall\, p \leq P-1$ by assumption.

An illustration of Case (ii) is given in Figure 8. ◀

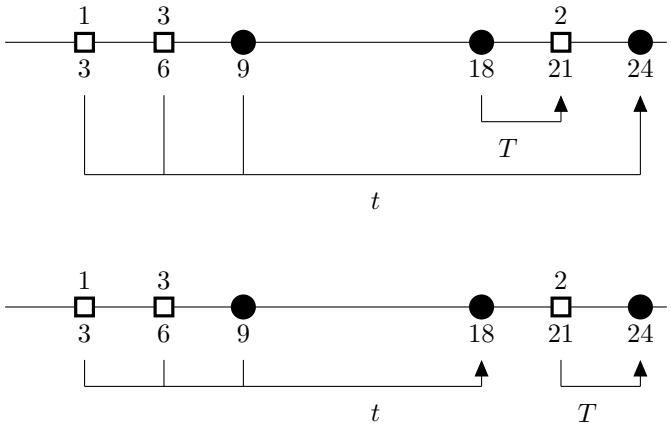

Figure 8 Top: a solution violating Property 8. Bottom: A non-worse solution complying with Property 8.

▶ **Property 9.** *Given three pickups p', p'' and p''' such that $\pi_{p'} \geq \pi_{p''} \geq \pi_{p'''}$ and two trips, t' of capacity 1 and t'' of capacity 2, such that $\pi_{p''} > \max\{c_{t'}^D, c_{t''}^D\}$, it is not optimal to assign p' or p'' to t' and the other two pickups to t''.*

Proof. Consider a solution in which p' is assigned to t' and p'' and p''' are assigned to t''. Re-assigning p''' to t' and p' to t'', the cost of t' decreases by $\pi_{p'} - \max\{\pi_{p'''}, c_{t'}^D\}$, while the cost of t'' increases by $\pi_{p'} - \pi_{p''}$. Therefore the cost of the solution decreases by $\pi_{p''} - \max\{\pi_{p'''}, c_{t'}^D\}$, which is non-negative because $\pi_{p''} \geq \pi_{p'''}$ and $\pi_{p''} < c_{t'}^D$ by assumption.

Similarly, consider a solution in which p'' is assigned to t' and p' and p''' are assigned to t''. Re-assigning p''' to t' and p'' to t'', may decrease the cost of t' but cannot increase the cost of t'' because $\pi_{p'} \geq \pi_{p''}$ by assumption. Therefore the cost of the solution does not increase.

The former case is illustrated in Figure 9. ◀

3 Dynamic programming algorithms

Hereafter we consider a relaxation of problem $2/1/PD/F$, namely problem $2/1/D/F$, according to taxonomy by Barbato et al. [1]. It is the same problem as $2/1/PD/F$ but without pickup items. In the remainder we recall a

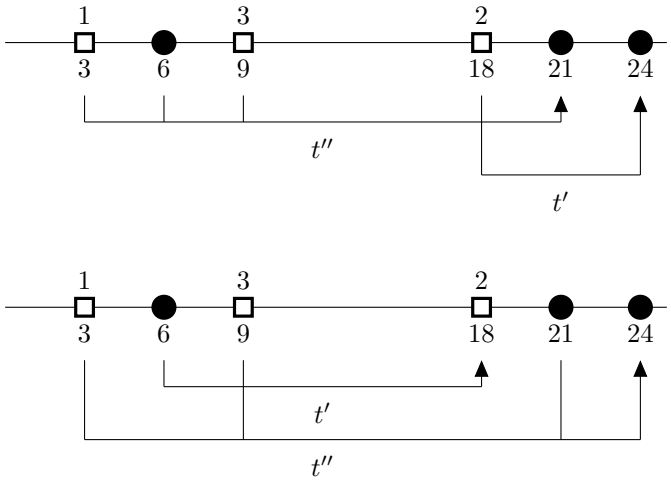

Figure 9 Top: a solution violating Property 9. Bottom: A non-worse solution complying with Property 9.

polynomial-complexity dynamic programming algorithm for problem $2/1/D/F$ and we describe a more complex version of the same idea to solve $2/1/PD/F$ with an exponential-complexity dynamic programming algorithm.

3.1 Dynamic programming for $2/1/D/F$

As shown by Barbato et al. [1], the $2/1/D/F$ problem can be reformulated as a shortest path problem on a suitably defined weighted acyclic digraph, where the nodes are numbered according to the given sequence of deliveries.

The dynamic programming state consists of the information on the last reached node, the cost of the partial path reaching the node, and its predecessor; a state S' dominates another state S'' if they reach the same node and the cost of S' is smaller than the cost of S''.

The cost w_{ij} associated with each arc (i, j) with $i < j$ is defined as follows:

$$w_{ij} = \begin{cases} \delta_j & \text{if } j = i+1 \\ \max\{\delta_j, \delta_{j-1}\} & \text{if } j = i+2 \\ \max\{\delta_{i+1}, \delta_j\} + \sum_{k=i+2}^{j-1} \delta_k & \text{if } j > i+2. \end{cases}$$

When $j = i + 1$, item j is delivered in a trip, with no other item; hence the trip cost is δ_j. When $j = i + 2$, items $j - 1$ and j are delivered in the same trip; hence, the trip cost is the maximum of the two distances δ_j and δ_{j-1}. When $j > i + 2$, item $i + 1$ is kept on board while all the next items are delivered one by one; finally it is delivered in a trip with item j; the cost of this decision is the sum of the cost of the trip with items $i + 1$ and j, i.e. $\max\{\delta_{i+1}, \delta_j\}$ and the costs of the trips delivering a single item at a time between item $i + 1$ and item j, i.e. $\sum_{k=i+2}^{j-1} \delta_k$.

An example is shown in Figure 10 and in Table 1.

Table 1 The weight matrix of the acyclic digraph corresponding to the small instance represented in Figure 10.

	1	2	3	4	5	6	7
0	11	57	75	129	168	227	238
1		57	57	75	129	170	227
2			18	54	93	152	170
3				54	54	98	152
4					39	59	98
5						59	59
6							9

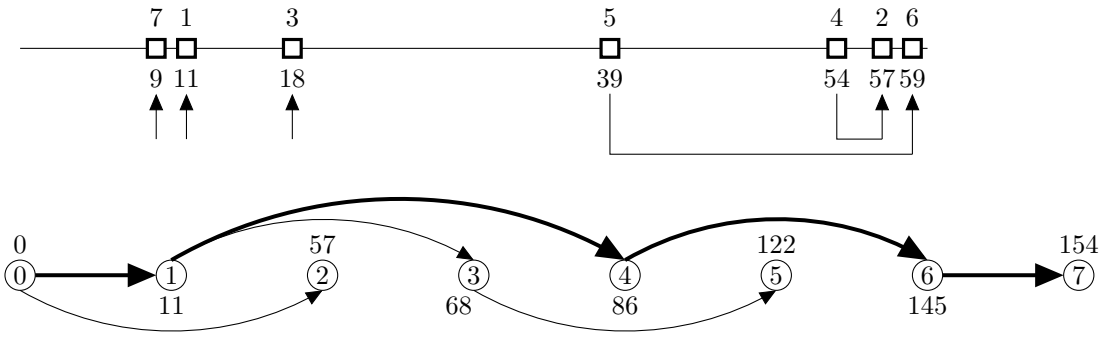

Figure 10 Top: a small instance of $2/1/D/F$ and its optimal trips. Bottom: the dynamic programming digraph: only arcs representing optimal predecessors of each state are drawn, together with the cost of the partial path reaching the corresponding node; bolded arcs correspond to the optimal solution.

The shortest path on an acyclic weighted digraph can be computed with $O(m)$ time-complexity, being m the number of arcs. The digraph defined above has a node for each delivery and an arc (i, j) for each pair of distinct nodes with $i < j$. Therefore, if D is the number of deliveries, the complexity of this algorithm is $O(D^2)$.

This algorithm provides an efficient way to compute a lower bound for $2/1/PD/F$ by simply disregarding the presence of pickups.

3.2 Dynamic programming for $2/1/PD/F$

The same idea of scanning the sequence of deliveries according to their arrival order can be adapted to design a dynamic programming algorithm for $2/1/PD/F$, with the additional difficulty that each arc (i, j) of the weighted acyclic digraph should be represented by as many distinct copies (with different cost) as the number of pickups that can determine the cost of the trip visiting deliveries $i + 1$ and j. Furthermore, when $i + 1$ and j are not consecutive, the same should also be done for the pickups associated with the single capacity trips visiting deliveries k with $i + 1 < k < j$. This yields a combinatorial explosion in the number of states.

In the dynamic programming algorithm we have devised for $2/1/PD/F$, the state $(i, j, \overline{P})$ represents the last reached node i of the graph (i.e. the last delivery) that has been assigned to trips, the delivery j on board of the shuttle, if any, and the subset $\overline{P}$ of pickups already assigned to trips. Then, the number of states is $O(D^2 2^P)$.

In constructing the dynamic programming acyclic weighted digraph, we distinguish two types of states: those with $j = 0$ (no delivery item is on board) and those with $j \neq 0$ (a delivery item j is on board).

Each state $(i, j, \overline{P})$ with $j \neq 0$ has $O(P)$ leaving arcs. Some leaving arcs correspond to completing an A-trip carrying the delivery items j and $i+1$. The head of each arc is a state $(i+1, 0, \overline{P'})$ with $\overline{P'} \supseteq \overline{P}$ and $|\overline{P'}| \leq |\overline{P}|+2$. Other leaving arcs correspond to completing a C-trip carrying the delivery item $i + 1$. The head of each arc is a state $(i + 1, j, \overline{P'})$ with $|\overline{P'}| \leq |\overline{P}| + 1$.

Each state $(i, 0, \overline{P})$ has $O(P)$ leaving arcs, too. Some leaving arcs correspond to completing a B-trip carrying the delivery item $i + 1$. The head of each arc is a state $(i + 1, 0, \overline{P'})$ with $\overline{P'} \supseteq \overline{P}$ and $|\overline{P'}| \leq |\overline{P}| + 2$. Other leaving arcs correspond to starting an A-trip carrying the delivery item $i + 1$. In this case the head of each arc is a state $(i + 1, i + 1, \overline{P})$.

Hence, the number of arcs is $O(PD^2 2^P)$ and this is also the worst-case time complexity of the dynamic programming algorithm.

4 Branch-and-bound algorithm 1

In this section we describe a branch-and-bound algorithm, where the sequence of branching decisions depends on the distance of deliveries from the origin instead of their position in the arrival sequence.

4.1 Branching policy

Relying on the properties outlined in Section 2, we devised the following branching policy.

Deliveries are considered by non-increasing distance from the origin and three types of decision are taken for each of them: (i) the type of trip which the delivery must be assigned to; (ii) the second delivery to be assigned to the same trip, if the trip capacity is 2; (iii) the pickup(s) to be assigned to the same trip.

As shown in the remainder, the properties outlined in Section 2 allow for always making decision (i) in a unique optimal way without branching, while decisions (ii) and (iii) may require branching.

When the farthest non-assigned delivery $j \in \mathcal{D}$ is considered for branching, we distinguish three cases

- Case I: deliveries $j-1$ and $j+1$ either do not exist (because $j=1$ or $j=D$, respectively) or have already been assigned to distinct trips;
- Case II: delivery j is included between two deliveries j' and j'' assigned to the same trip (i.e. $j' < j < j''$);
- Case III: otherwise.

Hereafter, for each case we outline the optimal decisions and the possible need for branching.

Case I. If deliveries $j-1$ and $j+1$ either do not exist or have already been assigned to distinct trips, then j cannot be matched with any other delivery, because of Property 2. It cannot even be assigned to a C-trip, because of Property 1. Hence, it must be assigned to a B-trip.

No decision has to be taken on the second delivery, since B-trips visit only one delivery.

Since the farthest trip in the current sub-problem is a B-trip with capacity 2, by Properties 6 and 7 it is optimal to assign the two farthest pickups to the trip. Hence, no branching is needed in this case, since all optimal decisions are unique.

Case II. If delivery j is included between two deliveries j' and j'' assigned to a same trip (i.e. $j' < j < j''$), then j must be assigned to a C-trip, because of Property 1. By Property 1 and Property 5, this can happen only when an A-trip is generated including non-consecutive deliveries, say $j' \in \mathcal{D}$ and $j'' \in \mathcal{D}$ with $j'' > j' + 1$. When this occurs all deliveries between j' and j'' are marked as "included", since they must be assigned to trips with capacity 1. This allows to immediately detect when the farthest delivery in a sub-problem, must be assigned to a C-trip.

In this case, no decision has to be taken on the second delivery, since C-trips visit only one delivery.

However, branching may be needed to decide the assignment of pickups to the C-trip. Let θ be the number of unassigned pickups that are farther than delivery j. Owing to Properties 6 and 8, if $\theta \leq 1$, then the farthest unassigned pickup, if one exists, is assigned to the C-trip and no branching is needed; otherwise, by Property 9, branching occurs: $\lfloor \theta/2 \rfloor + 1$ sub-problems are generated; for each sub-problem $k = 0, \ldots, \lfloor \theta/2 \rfloor$, the $2k$ farthest pickups are skipped and the next farthest unassigned pickup, if any, is assigned to the C-trip.

By Property 9, once some pickups have been skipped, they cannot be later assigned to trips of capacity 1. Therefore, the branching policy described above is made more restrictive and efficient by forbidding some sub-optimal branches. Skipped pickups are marked as "skipped" and when Case II occurs at some later stage the computation of θ and the consequent generation of sub-problems disregards all pickups marked as "skipped".

Case III. For Properties 3 and 4, the farthest delivery order j is assigned to a B-trip only when this is mandatory, because of previous branching decisions (Case I). For Property 1, it is sub-optimal to assign delivery j to a C-trip. Therefore, j is assigned to an A-trip.

Branching occurs on the second delivery to be assigned the same trip and for each possible choice a distinct sub-problem is generated. By Property 2, every time a trip of capacity 2 is generated by the branching procedure the set $\mathcal{D}$ is partitioned into intervals that can be treated as independent delivery subsets. Hence, the search for candidates to be assigned to the new A-trip is limited to the delivery subset which the farthest delivery j belongs to. We indicate it with $\overline{\mathcal{D}}$.

For Property 5, the only candidates that need to be considered are the deliveries $j' \in \overline{\mathcal{D}}$ with $j' < j$ such that $\delta_{j'} > \delta_k \ \ \forall\, k = j'+1, \ldots, j-1$ and the deliveries $j'' \in \overline{\mathcal{D}}$ with $j'' > j$ such that $\delta_{j''} > \delta_k \ \ \forall\, k = j+1, \ldots, j''-1$.

Owing to Properties 6 and 7, no branching on pickups assignment is needed in this case: the two farthest pickups, if any, are assigned to the A-trip in all sub-problems.

Final assignments. In case no deliveries are left and some pickups remain unassigned, it is trivially optimal to assign them in pairs to D-trips, according to their order starting from the farthest ones. Hence, this final step does not require branching.

An example. Figure 11 shows three different instances. For all of them, it shows the corresponding optimal solutions.

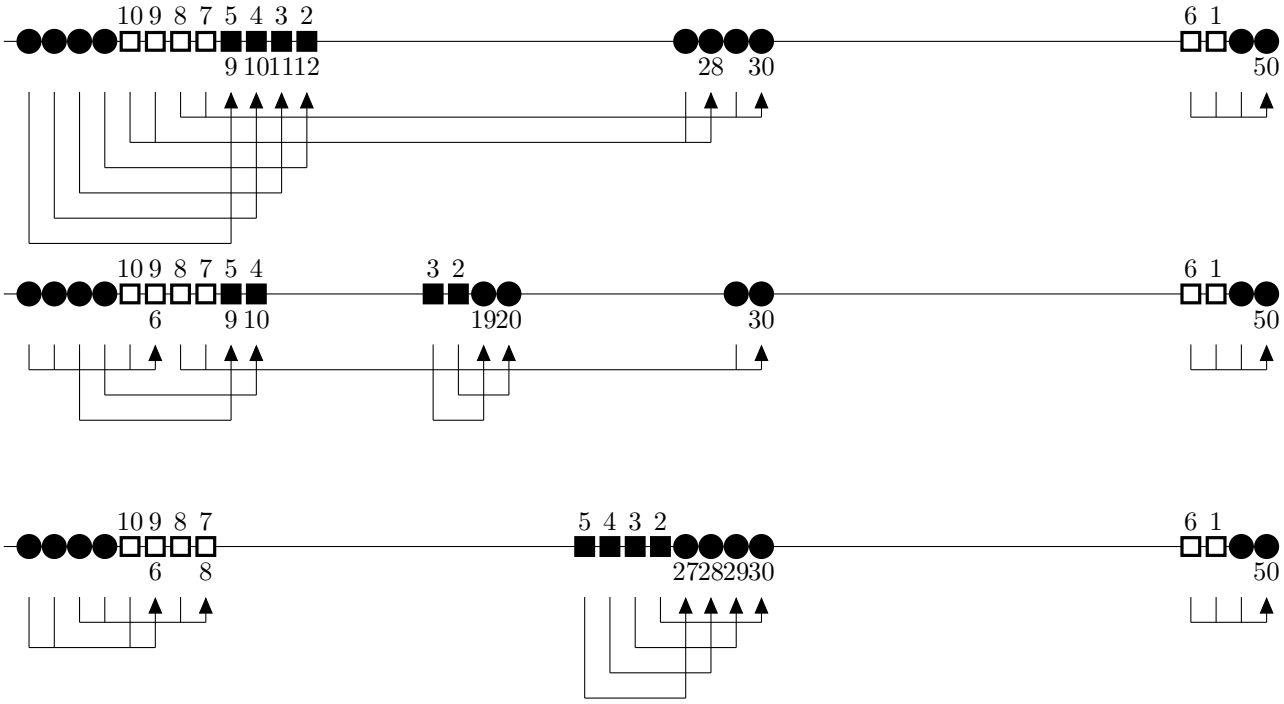

Figure 11 Examples of different optimal solutions depending on the distances of pickups and deliveries. Dots indicate pickups, squares indicate deliveries, whose numbering is reported above the squares. White squares are deliveries assigned to trips with capacity 2, while black squares are deliveries assigned to trips with capacity 1. Below the line, optimal trips are represented: for each trip, an arrow indicates the farthest location to be visited.

The composition of optimal trips is different according to the distances of pickups and deliveries, although their order is the same. Consider the sub-problem in which delivery 1 has been matched with delivery 6 in a double capacity trip. Hence, deliveries 2 to 5 are marked as included and must be assigned to single capacity trips. When the next delivery, i.e. delivery 2 is considered and is assigned a C-trip, different branches are generated in the branch-and-bound algorithm and the optimal branch is different in the three instances. In the top part of the figure, the optimal branch is when two pairs of pickups are skipped. In the center of the figure, the optimal branch is when one pair of pickups is skipped. In the bottom part of the figure, the optimal branch is when no pair of pickups is skipped.

The full branching tree related to the instance shown in Figure 11 is reported in the appendix.

4.2 Lower bounding

In each sub-problem generated by the branching policy, the cost of the partial solution is updated. Thanks to the sequence in which deliveries are considered, and thanks to branching on possibly optimal assignments of pickups to trips, the cost of each trip t generated by branching can be immediately determined.

Besides the cost of the partial solution, a lower bound on the cost of visiting the unassigned deliveries and pickups is computed. If the cost of the partial solution increased by the lower bound exceeds the value of an upper bound, than the sub-problem is fathomed. The computation of a suitable upper bound is dealt with in Subsection 4.3.

Our branch-and-bound algorithm employs two distinct lower bounding sub-routines that do not dominate each other. Both of them have polynomial complexity and they are very fast in practice. For each sub-problem in the branch-and-bound tree both lower bounding procedures are executed and the best of the two lower bounds is retained.

4.2.1 Lower bound 1

The effect of our branching strategy is to partition the sequence of deliveries into disjoint subsequences separated by the already assigned deliveries. For each subsequence of unassigned deliveries, a lower bound is computed by

disregarding the existence of pickups and solving the correspondent instance of $2/1/D/F$ to optimality with the dynamic programming algorithm outlined in Section 3.1. Furthermore, the lower bound corresponding to each delivery $k \in \mathcal{D}$ marked as "included" is given by δ_k. Summing up the lower bounds for subsequences of unassigned deliveries and the lower bounds for the "included" deliveries, a valid lower bound is obtained for the whole set of unassigned deliveries.

4.2.2 Lower bound 2

The second lower bounding technique consists of disregarding the arrival sequence of deliveries. When the constraint on the sequence is dropped, it is always optimal to form double capacity trips in each subsequence by grouping the unassigned deliveries in pairs starting from the farthest ones. This generates a set of double capacity trips with unassigned deliveries from the subsequences and a set of single capacity trips with the deliveries marked as "included". Finally, unassigned pickups can be optimally assigned to these trips, after sorting them, by filling their available capacity starting from the farthest ones. Possibly exceeding pickups are assigned to additional D-trips, as necessary.

4.3 Upper bounding

For each sub-problem in the branch-and-bound tree a valid upper bound, i.e. a feasible complete solution, is computed by a greedy procedure. After computing lower bound 1 by disregarding unassigned pickups, a set $\mathcal{T}'$ of trips with only deliveries is obtained. Then, all unassigned pickups are assigned to the trips in $\mathcal{T}'$ following the same greedy rule used to compute lower bound 2, i.e. iteratively filling the capacity of the farthest trip in $\mathcal{T}'$ with the farthest unassigned pickups. Possibly exceeding pickups are assigned to additional D-trips, as necessary.

5 Branch-and-bound algorithm 2

By suitably combining ideas from the dynamic programming algorithm described in Section 3 and the branch-and-bound algorithm described in Section 4, we devised another branch-and-bound algorithm, where delivery items are considered according to their given arrival sequence.

5.1 Branching policy

When a new delivery item j arrives, the following actions can be taken: (a) item j is kept on board the shuttle (no trip); (b) only item j is delivered (a B-trip or a C-trip is generated); (c) item j and another delivery item $i < j$ are delivered (an A-trip is generated). Case (a) is possible only if the shuttle is empty when j arrives; case (c) is possible only if the shuttle is not empty when j arrives; case (b) is possible only if the shuttle contains no item $i < j$ with $\delta_i \leq \delta_j$, when j arrives.

In case (a) no branching is needed: the state of the shuttle is updated and the next delivery item is considered. In cases (b) and (c), the trip is completed with one or two pick-up items, according to the available capacity of the shuttle. If two pick-up items can be accommodated on it (A-trip or B-trip), they are chosen in consecutive positions, owing to Property 7. The branching policy generates a new sub-problem for each possible choice of the pick-up items(s) in the trip. All unassigned pick-up items are considered, starting from the farthest one. The last farthest pick-up item considered (i.e. the only one if the trip capacity is 1 or the farthest among the two consecutive pick-up items if the trip capacity is 2) is the farthest pick-up item among those that are not farther than the delivery item j, i.e.

$$i^* = \arg \max_{i \in P : \pi_i \leq \delta_j} \{\pi_i\}.$$

All pick-up items closer than i^* need not be examined, by Property 6.

Final assignments. As with the branch-and-bound algorithm described in Section 4, if some pickups are left unassigned, they are assigned in consecutive pairs to D-trips, starting from the farthest ones in order to obtain a complete solution.

5.2 Bounding

The cost of the trip generated by branching can be immediately determined and added to the cost of the partial solution that corresponds to each sub-problem.

Then the same lower bounding algorithms described in Subsections 4.2.1 and 4.2.2 are used. The former one disregards pick-up items and solves an instance of $2/1/D/F$ to optimality by dynamic programming by considering the remaining delivery items. The latter one disregards the arrival sequence of the remaining deliveries, grouping them in pairs from the farthest ones and then optimally assigning the remaining pick-up items to the resulting trips.

Also upper bounds are computed as described Subsection 4.3.

6 A mixed-integer linear programming model

In this section we present a binary linear programming model to solve problem $2/1/PD/F$. It can be solved by general-purpose MILP solvers and it serves as a term of comparison for the specialized algorithms described in the previous sections.

First, we reformulate problem $2/1/PD/F$ as a shortest path problem on a suitably defined acyclic, capacitated, weighted, layered digraph. Then we present the corresponding ILP model.

Let $\mathcal{T}$ be an indexed set of T trips, large enough to allow for a feasible and optimal solution accommodating all pickups and deliveries.

The nodes of the digraph are partitioned into $T+1$ layers, numbered from 0 to T. Layers from $t=0$ up to $t=D-1$ represent the state of the shuttle when delivery $t+1$ arrives. Additional layers, if any, are needed in case the number of pickups exceeds the total capacity of the trips up to layer D.

The nodes in each layer t represent the state of the shuttle after the first t trips/no-actions have been executed (see next paragraph). Each layer $t \leq D-1$ contains as many nodes as the possible delivery items on board of the shuttle, i.e. delivery items already received but not yet delivered. Each layer also contains a node corresponding to having the shuttle empty. We indicate by (t,j) the node of layer t with delivery j on the shuttle and by $(t,\emptyset)$ the node of layer t with empty shuttle.

Arcs from layer $t-1$ to layer t may correspond to feasible trips as well as to no-action, according to the following classification.

- No-action arcs go from nodes $(t-1,\emptyset)$ to nodes (t,t); their delivery cost is null and their capacity is null, since they do not correspond to any trip.
- A-arcs, i.e. arcs corresponding to A-trips, go from nodes $(t-1,j)$ to nodes $(t,\emptyset)$; their delivery cost is $\max\{\delta_j,\delta_t\}$ and their capacity is equal to 2.
- B-arcs, i.e. arcs corresponding to B-trips, go from nodes $(t-1,\emptyset)$ to nodes $(t,\emptyset)$; their delivery cost is δ_t and their capacity is equal to 2.
- C-arcs, i.e. arcs corresponding to C-trips, go from nodes $(t-1,j)$ to nodes (t,j); their delivery cost is δ_t and their capacity is equal to 1.
- D-arcs, i.e. arcs corresponding to D-trips, go from nodes $(t-1,\emptyset)$ to nodes $(t,\emptyset)$ if and only if $t>D$; their delivery cost is null and their capacity is equal to 2.

A path from $(0,\emptyset)$ to node $(T,\emptyset)$ represents an assignment of deliveries to trips. Pickups must also be assigned to the selected trips according to the capacities of the chosen arcs.

Properties outlined in Section 2 can be used to reduce the digraph. For instance, from a node (t,i) with $i<t$ a C-arc is inserted to node $(t+1,i)$ only if $\delta_{t+1}<\delta_i$ owing to Property 5.

An example of this construction is shown in Figure 12 for a toy instance with 7 deliveries (pickups are not indicated).

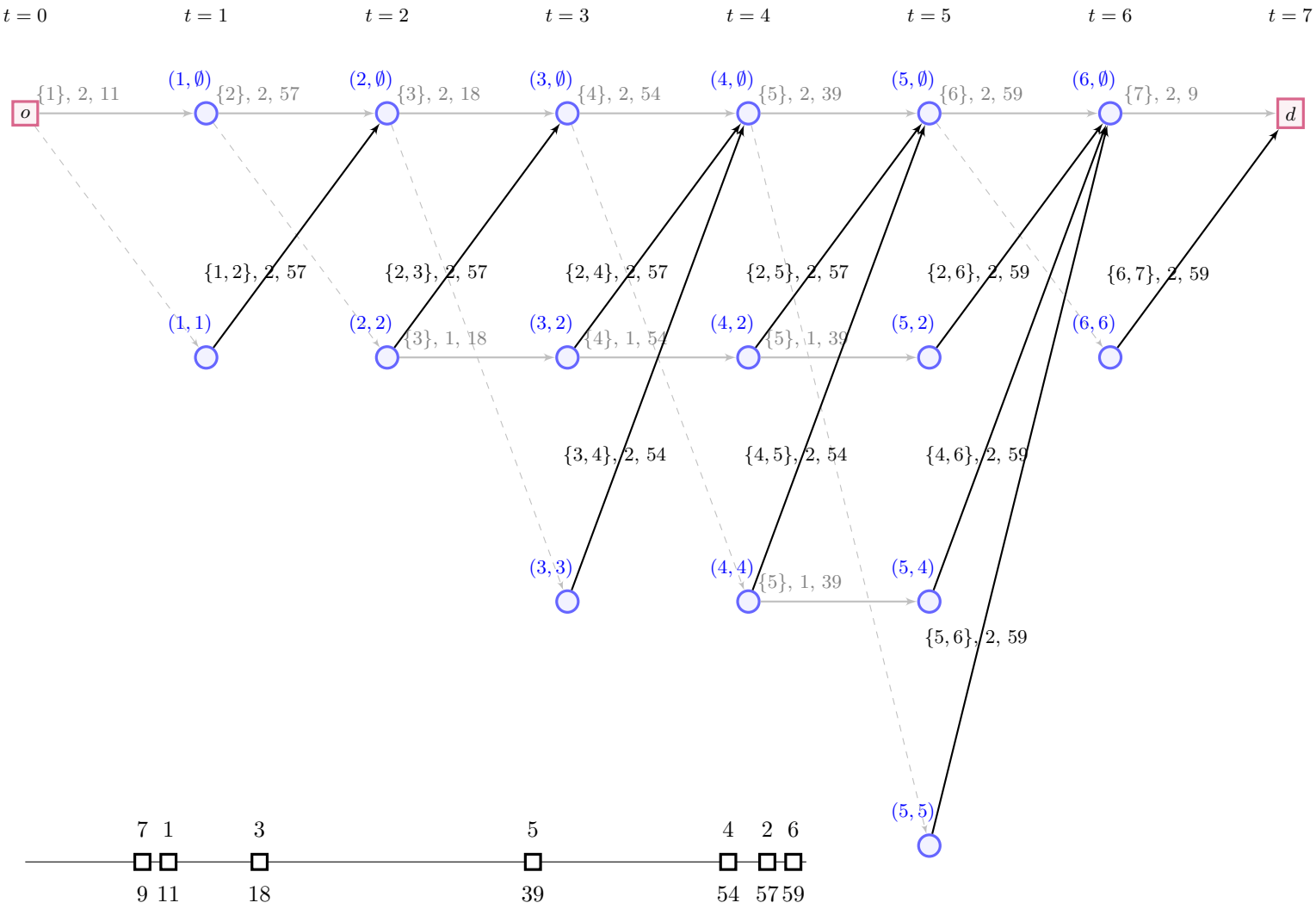

Figure 12 Graph sample for $D = \{1, \dots, 7\}$ and distances $\{11, 57, 18, 54, 39, 59, 9\}$ (see the bottom part of the figure). For each arc a corresponding to a feasible trip, we report the set of deliveries to perform, the capacity q_a, and the delivery cost w_a. Dashed arcs correspond to no-action, with null delivery cost and null capacity.

The search for a feasible and optimal path on the digraph defined above can be translated into an MILP model, as follows.

Data. We indicate by N^t the subset of nodes in layer $t \in \mathcal{T} \cup \{0\}$. We indicate by $\Delta^-(t,n)$ and $\Delta^+(t,n)$ the in-star and the out-star of any node $n \in N^t$ for $t \in \mathcal{T} \cup \{0\}$. We indicate by A^t the subset of arcs from nodes of layer $t-1$ to nodes of layer t forall $t \in T$. We indicate by w_a the delivery cost and by q_a the capacity of each arc a in the digraph. We assume that pickup items are numbered in non-increasing order according to their distance from the origin.

Variables. We use the following variables:

- $x_a \in \{0,1\}$ is equal to 1 if and only if arc a is traversed, 0 otherwise;
- $y_p^t \in \{0,1\}$ is equal to 1 if and only if pickup $p \in \mathcal{P}$ is assigned to trip $t \in \mathcal{T}$;
- $z_p^t \in \{0,1\}$ is equal to 1 if and only if pickups p and $p-1$, with $p \in \mathcal{P} \setminus \{1\}$, are assigned to trip $t \in \mathcal{T}$;
- $c^t \geq 0$ is the cost of trip $t \in T$.

The model is as follows.

$$\min \quad \sum_{t \in \mathcal{T}} c^t \tag{1a}$$

$$s.t. \quad \sum_{a \in A^1} x_a = 1 \tag{1b}$$

$$\sum_{a \in \Delta^-(t,n)} x_a = \sum_{a \in \Delta^+(t,n)} x_a \qquad \forall\, t \in \mathcal{T}\backslash\{1,T\},\ \forall\, n \in N^t \tag{1c}$$

$$\sum_{p \in \mathcal{P}} y_p^t + \sum_{p \in \mathcal{P}\setminus\{1\}} z_p^t \leq 1 - \sum_{a \in A^t : q_a = 0} x_a \qquad \forall\, t \in \mathcal{T} \tag{1d}$$

$$\sum_{p \in \mathcal{P}\setminus\{1\}} z_p^t \leq 1 - \sum_{a \in A^t : q_a = 1} x_a \qquad \forall\, t \in \mathcal{T} \tag{1e}$$

$$\sum_{p \in P} y_p^t + \sum_{p \in \mathcal{P}\setminus\{1\}} z_p^t \leq 1 \qquad \forall\, t \in \mathcal{T} \tag{1f}$$

$$\sum_{t \in \mathcal{T}} y_1^t + z_2^t = 1 \tag{1g}$$

$$\sum_{t \in \mathcal{T}} (y_p^t + z_p^t + z_{p+1}^t) = 1 \qquad \forall\, p \in \{2, \ldots, P-1\} \tag{1h}$$

$$\sum_{t \in \mathcal{T}} y_P^t + z_P^t = 1 \tag{1i}$$

$$\sum_{a \in A^t} w_a x_a \leq c^t \qquad \forall\, t \in \mathcal{T} \tag{1j}$$

$$\sum_{p \in \mathcal{P}} \pi_p y_p^t + \sum_{p \in \mathcal{P}\setminus\{1\}} \pi_{p-1} z_p^t \leq c^t \qquad \forall\, t \in \mathcal{T} \tag{1k}$$

$$c^t \geq c^{t+1} \qquad \forall\, t = D+1, \ldots, T \tag{1l}$$

$$x_a \in \{0,1\} \qquad \forall\, t \in \mathcal{T}, \forall\, a \in A^t \tag{1m}$$

$$y_p^t \in \{0,1\} \qquad \forall\, p \in \mathcal{P}, \forall\, t \in \mathcal{T} \tag{1n}$$

$$z_p^t \in \{0,1\} \qquad \forall\, p \in \mathcal{P}\backslash\{1\},\ \forall\, t \in \mathcal{T} \tag{1o}$$

$$c^t \geq 0 \qquad \forall\, t \in \mathcal{T} \tag{1p}$$

Flow conservation constraints (1b) and (1c) ensure that exactly one arc is traversed in each layer of the digraph. For arcs corresponding to no-action, the assignment of pickups is forbidden by constraints (1d). For C-arcs corresponding to trips with capacity 1, only the assignment of a single pickup is allowed, owing to constraints (1e). For arcs corresponding to trips with capacity 2, the assignment of at most two pickups is ensured by constraints (1f). Constraints (1f) also apply to D-trips. Assignment constraints (1g), (1h) and (1i) force the assignment of each pickup to a trip. Constraints (1g) and (1i) are just special cases of (1h) for the first and the last pickup. Constraints (1j) and (1k) define the cost of each trip, setting two lower bounds to it, the former due to the delivery cost, the latter to the pickup cost. To the aim of restricting symmetries in

the solution space, constraints (1l) impose that the costs of the final D-trips be sorted in non-increasing order. Finally, constraints (1m)–(1p) define the domain of the variables.

7 Computational results

To comparatively evaluate the effectiveness of the different algorithms we have devised for the $2/1/PD/F$ problem, we generated a set of random instances as follows: the number n of pickup and deliveries was set to a value ranging from 10 to 50 by steps of 5. In all instances the number pickup and deliveries was the same. The distances of pickups and deliveries from the origin were generated in two different ways: in "uniform" instances they were drawn at random from a uniform probability distribution in the range $[1, 2n]$; in "Gaussian" instances they were drawn at random from a Gaussian distribution whose mean μ and standard deviation σ were set at $\mu = n/2$ and $\sigma = n/6$ respectively. The extraction of integer values was done in the range $[1, 2n]$, which means 3σ were allowed on the left ot the mean value and 9σ on the right. All random values were directly generated as integers. For each type of instance and each value of n, 512 instances were generated.

Each instance was solved with the dynamic programming algorithm described in Section 3 (DP), with the branch-and-bound algorithm described in Section 4 (BB1), with the branch-and-bound algorithm described in Section 5 (BB2) and with two state-of-the-art MILP solvers, namely Cplex and Gurobi. In particular, we used IBM ILOG Cplex Interactive Optimizer 22.1.0.0 and Gurobi Optimizer version 11.0.1 build v11.0.1rc0.

All tests were done on a PC equipped with Intel Core i7-6700K 4.00GHz (4 cores, 8 threads) and 64 GB DDR4 2133 MHz RAM.

The results (computing time expressed in seconds) are reported in Tables 2 and 3 for uniform and Gaussian instances, respectively. Table 2 also reports the number of rows and columns in the MILP model for each value of n. Empty cells correspond to instance size not solved within a time limit of ten minutes.

Table 2 Computing time to solve instances generated with uniform distribution (^{a}: 511 instances solved).

n	rows	cols	DP	BB1	BB2	Cplex	GuRoBi
10	102	352	0.52	0.02	0.05	0.07	0.05
15	150	759	94.23	0.03	0.14	0.18	0.14
20	202	1300		0.04	8.36	0.64	0.31
25	278	2063		0.07		1.85	0.82
30	321	2874		0.12		4.97	1.28
35	391	3943		0.19		9.58^a	1.93
40	432	5036		0.35			2.55
45	515	6439		1.06			4.19^a
50	572	7858					

Table 3 Computing time to solve instances generated with Gaussian distribution (^{a}: 502 instances solved; ^{b}: 507 instances solved).

n	DP	BB1	BB2	Cplex	GuRoBi
10	0.52	0.02	0.04	0.06	0.04
15	94.39	0.03	0.22	0.22	0.15
20		0.05	35.14^a	0.69	0.33
25		0.13		1.75	0.67
30		0.31		3.60	1.14
35		0.99		7.62	1.99
40		2.29			2.53
45		11.60^b			3.92
50					

As expected, dynamic programming is not competitive with the other methods. BB1 outperforms BB2 with a very small number of exceptions. Average computing times of GuRoBi always dominate those of Cplex. BB1 outperforms MILP solvers especially for relatively small instances and for uniform instances of all sizes, but its

computing time tends to suddenly explode for n around 45 on Gaussian instances. In general, Gaussian instances look slightly more difficult than uniform instances for branch-and-bound, while ILP solvers are more robust with respect to this feature.

We also evaluated the effectiveness of $LB1$ and $LB2$ separately in BB1. The results are summarized in Table 4. For each instance type and size, the table reports the average number of sub-problems generated in the branch-and-bound tree besides the root node and (in parentheses) the number of instances solved within 5 seconds, when both lower bounding techniques were used and when only $LB1$ and only $LB2$ were used.

Table 4 Column *Both* indicates the average number of sub-problems in the branch-and-bound tree when all instances are solved to proven optimality. Columns $LB1$ and $LB2$ indicate the average number of sub-problems generated in the instances solved within five seconds and their number, when only one of the two lower bounds is used.

	Uniform			Gaussian		
n	*Both*	$LB1$	$LB2$	*Both*	$LB1$	$LB2$
10	17.96	86.29 (512)	41.00 (512)	37.39	232.42 (512)	47.44 (512)
15	285.78	1379.55 (512)	526.49 (512)	298.99	12144.84 (511)	422.07 (512)
20	742.21	2765.35 (512)	2677.64 (512)	1229.22	18919.54 (508)	2744.53 (511)
25	1801.28	10133.30 (508)	14630.55 (512)	3954.66	60838.90 (488)	14128.07 (510)
30	5172.96	18539.14 (503)	91868.96 (508)	16016.58	60967.11 (469)	87275.92 (475)
35	12106.22	41357.19 (501)	146502.07 (131)	102633.93	95764.38 (435)	139326.53 (136)
40	34661.41	37053.06 (496)	231044.33 (3)	293449.85	123908.19 (418)	153565.17 (6)
45	130280.93	37534.52 (485)	(0)	2764951.38	148082.07 (384)	12150.00 (1)

From the analysis of the computational results it is apparent that $LB2$ is more effective when n is relatively small while $LB1$ is more effective for large n. For increasing values of n, $LB2$ alone quickly becomes ineffective; however, the difference between using $LB1$ alone and using both lower bounds remains significant, showing an effective complementarity between the two bounding techniques. These observations apply to both types of instances.

Table 5 shows the integrality gap yielded by the ILP model illustrated in Section 6. The optimal value of the continuous relaxation is within few percentage points from the discrete optimum and the integrality gap tends to slowly decrease when n increases.

Table 5 Optimal value of the discrete problem, optimal value of the continuous relaxation at the root node and percentage gap (average values for each group of 512 instances).

	Uniform			Gaussian		
n	opt.	relax	gap	opt.	relax	gap
10	70.66	68.23	3.44%	33.82	32.14	4.96%
15	151.92	147.53	2.89%	71.13	67.69	4.83%
20	263.30	256.19	2.70%	130.34	124.03	4.84%
25	402.81	392.53	2.55%	193.40	184.79	4.45%
30	573.37	559.73	2.38%	287.13	274.91	4.26%
35	771.76	755.68	2.08%	378.50	363.56	3.95%
40	998.38	979.14	1.93%	502.11	482.78	3.85%
45	1250.97	1228.90	1.76%	619.28	595.76	3.80%

8 Conclusions

The algorithms presented in this paper fill the last gap that had remained open in the classification proposed in Barbato et al. [1]. Possible extensions to slightly more complex cases can be now tackled. One natural extension is the $2/1/PD/V$ model, where the origin of the trips is located in the center of the rail and each pick-up or delivery order can be satisfied in one of two locations, one on each side of the origin.

Other extensions of practical interest concern the minimization of the energy consumed, instead of the total distance traveled. This is especially interesting in the case where the energy consumption of the shuttle depend on the weight that it carries while moving. A proxy of the weight can be assumed to be the number of boxes.

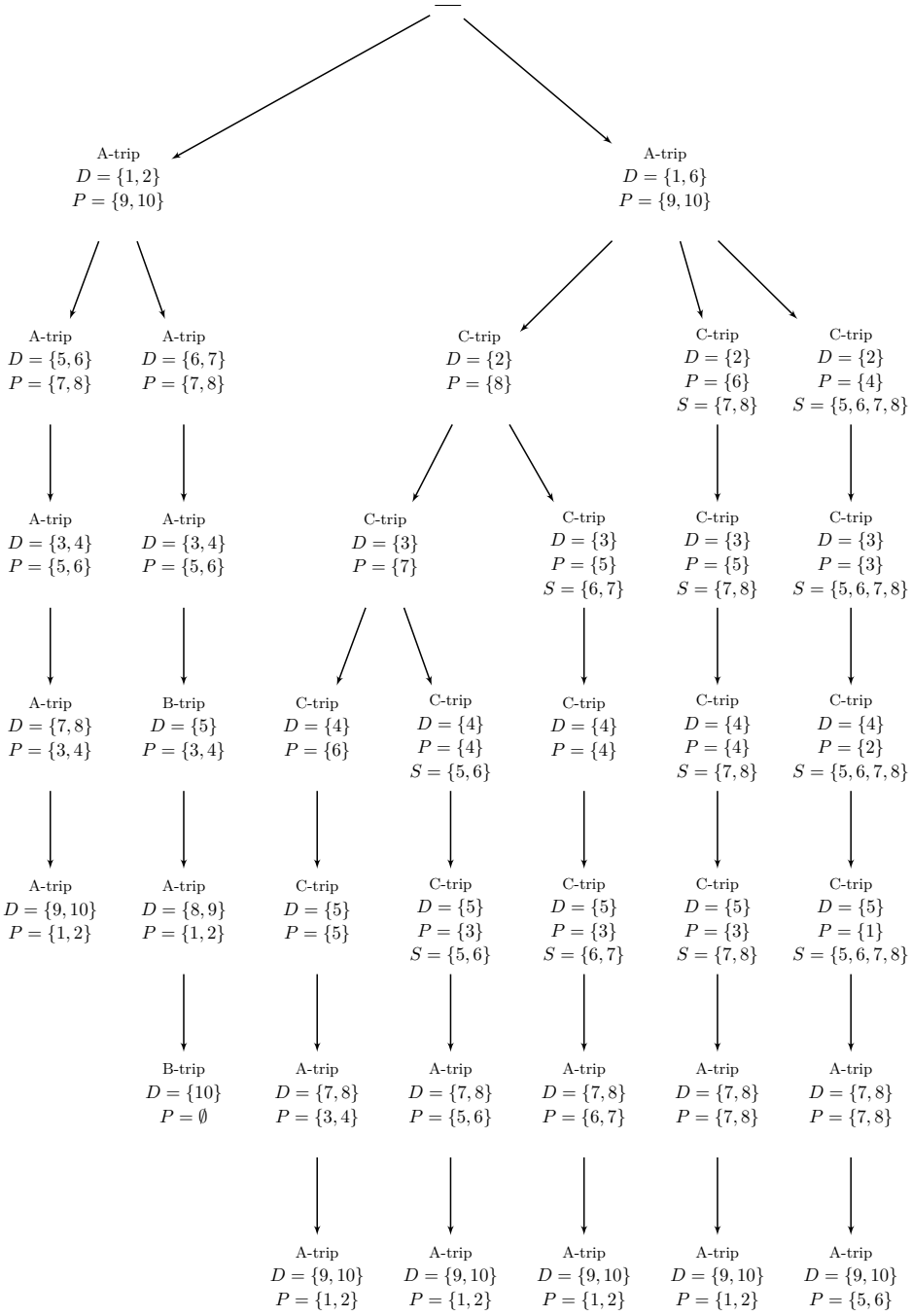

Figure 13 The branching tree corresponding to the instance in Figure 11. For each node, the type of trip in indicated first, followed by the delivery items (D) and the pickup items (P). In some nodes some pickup items (S) are marked as skipped. Branching produces 7 distinct solutions.

More complex versions that deserve investigation are on-line optimization and problems with deadlines or due dates for pick-up orders or release dates for delivery orders.

A An example of branching tree

In Figure 13 we show the full branching tree corresponding to the instance in Figure 11. Pickup items have been numbered from 1 to 10 by increasing distance from the origin.

References

1 Michele Barbato, Alberto Ceselli, and Giovanni Righini. Paths and Matchings in an Automated Warehouse. In M. Paolucci et al., editors, *Advances in Optimization and Decision Science for Society, Services and Enterprises*, volume 3 of *AIRO Springer Series*, pages 151–159. Springer, 2019.

2 Michele Barbato, Alberto Ceselli, and Giovanni Righini. A polynomial-time dynamic programming algorithm for an optimal picking problem in automated warehouses. *J. Sched.*, 27:393–407, 2024.

3 Nils Boysen, René de Koster, and Felix Weidinger. Warehousing in the e-commerce era: A survey. *Eur. J. Oper. Res.*, 277(2):396–411, 2019.

4 Nils Boysen and Konrad Stephan. A survey on single crane scheduling in automated storage/retrieval systems. *Eur. J. Oper. Res.*, 254(3):691–704, 2016.

5 René de Koster, Tho Le-Duc, and Kees Jan Roodbergen. Design and control of warehouse order picking: A literature review. *Eur. J. Oper. Res.*, 182(2):481–501, 2007.

6 Jinxiang Gu, Marc Goetschalckx, and Leon F. McGinnis. Research on warehouse operation: A comprehensive review. *Eur. J. Oper. Res.*, 177(1):1–21, 2007.

7 Kees Jan Roodbergen and Iris F. A. Vis. A survey of literature on automated storage and retrieval systems. *Eur. J. Oper. Res.*, 194(2):343–362, 2009.

8 J. P. van den Berg and W. H. M. Zijm. Models for warehouse management: Classification and examples. *Int. J. Prod. Econ.*, 59(1):519–528, 1999.