Synchronized Pickup and Delivery Problems with Connecting FIFO Stack

Michele Barbato, Alberto Ceselli, and Nicolas Facchinetti

Abstract In this paper we introduce a class of routing problems where pickups and deliveries need to be performed in two distinct regions, and must be synchronized by considering the presence of a first-in-first-out channel linking them. Our research is motivated by applications in the context of automated warehouses management. We formalize our problem, defining eight variants which depend on the characteristics of both the pickup and delivery vehicles, and the first-in-first-out linking channel. We show that all variants are in general **NP**-hard. We focus on two of these variants, proving that relevant sub-problems can be solved in polynomial-time. Our proofs are constructive, consisting of resolution algorithms. We show the applicability of our results by computational experiments on instances from the literature.

1 Introduction

In the field of logistics the analysis of routing problems plays a fundamental role: when applied to real-world situations, it may lead to considerable cost savings for transportation companies operating over large regions, see [8].

Routing problems are often encountered on a smaller scale too. As a relevant case, in this paper we introduce the family of *synchronized pickup and delivery problems with connecting first-in-first-out stack* (SPDP-FS). It is inspired by the

Michele Barbato
Università degli Studi di Milano, Dipartimento di Informatica, OptLab, Via Celoria 18 – 20133 Milan (MI), Italy, e-mail: michele.barbato@unimi.it

Alberto Ceselli
Università degli Studi di Milano, Dipartimento di Informatica, OptLab, Via Celoria 18 – 20133 Milan (MI), Italy, e-mail: alberto.ceselli@unimi.it

Nicolas Facchinetti
Università degli Studi di Milano, Dipartimento di Informatica, OptLab, Via Celoria 18 – 20133 Milan (MI), Italy, e-mail: nicolas.facchinetti@studenti.unimi.it

production system of an industrial partner on smart cosmetics manufacturing [1]. Overall, problems in such a setting concern the transportation of items between a pickup area and a delivery area inside the same factory. The two areas communicate through a first-in-first-out (FIFO) conveyor. As a consequence the pickup and the delivery routes cannot be independently optimized. Instead they must be coordinated so as not to violate the FIFO policy, and potentially synchronized. In particular, in the application inspiring this work, the pickup vehicle is an automated crane which collects items from the warehouse; at the end of each trip, that is a sequence of pickup operations in which the crane starts empty, moves reaching and incrementally loading items finally bringing them to an unloading spot, the items are put on the conveyor in the same order they have been collected. The delivery vehicle is an automated shuttle which must wait for loading until a batch of items is put on the conveyor, but then might be free to follow any order to deliver them. Since the vehicle capacities are finite and typically much lower than the overall number of items to pickup and deliver (*e.g.,* at most 2 or 3 items can be transported by a single trip) the resulting problem of optimizing the pickup and delivery routes while satisfying the constraints arising from the FIFO rule does not reduce in general to classical routing problem as, *e.g.,* the travelling salesman problem [2].

In this paper we provide a modeling of the above situation by introducing SPDP-FS variants, each depending on the degree of freedom the vehicles have when loading items to and from the conveyor with respect to the order of visits of the pickup and delivery locations.

Contributions and Outline. In Sect. 2 we formally define eight SPDP-FS variants and show that they are all in general **NP**-hard. We consider the case in which the capacities of the pickup and delivery vehicles are part of the input and study the cases in which they are fixed parameters. In Sect. 3 we focus on the variants in which the order of the items on the conveyor must coincide with both their pickup and delivery orders. We study how to complete partial solutions. To this aim we first present an algorithm to construct a conveyor order which is consistent with two given sets of pickup and delivery trips; our algorithm thus determines the feasibility of the given sets of trips. Next, we present a dynamic programming algorithm to determine the optimal pickup and delivery trips which are consistent with a given conveyor order. In order to evaluate the suitability of the latter algorithm as a subroutine for more sophisticated SPDP-FS heuristics, in Sect. 4 we test it on instances from the literature [7].

2 Modelling and Complexity

Throughout the paper, let $n \in \mathbb{Z}_{>0}$ and let $G = (V,A)$ be a complete digraph on the vertex set $V = \{0,1,2,\dots,n\}$. The *pickup network* is a weighted digraph $D^1 = (G,c^1)$ with arc cost function $c^1 : A \to \mathbb{R}_+$ and whose vertices in $V \setminus \{0\}$ represent the positions of n items identical in terms of size. Intuitively, D^1 represents

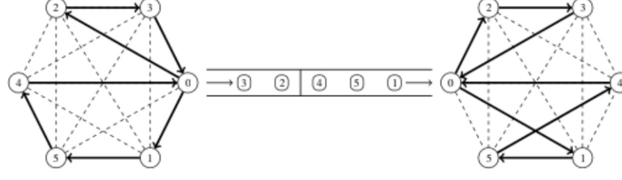

Fig. 1 Schematic illustration of the automated warehouse. The pickup area is on the left, delivery area on the right. Both vehicles are assumed to have capacity 3. No permutations and no overlaps are allowed.

a storage area (a warehouse in the real application) for raw materials demanded by the production units of a factory. The *delivery network* is represented by another weighted digraph $D^2 = (G, c^2)$ with arc cost function $c^2 : A \to \mathbb{R}_+$. Intuitively, D^2 represents a production area. We assume that the production unit located at vertex $i \in V \setminus \{0\}$ of D^2 must receive the item located at vertex $i \in V \setminus \{0\}$ of D^1; in other words, items and production units are in one-to-one correspondence.

The items are collected by a vehicle of finite capacity k_1 by visiting the corresponding vertices of D^1. The capacity is finite, and typically lower than $|V \setminus \{0\}|$: in order to pick up all items, the vehicle performs several trips, where a *trip* is a sequence of vertices forming a simple directed cycle starting and ending at vertex 0. Once items have been collected, they are delivered in an analogous manner in D^2 by a second vehicle of capacity k_2.

Overall, a solution (P, D) consists of an ordered sequence of pickup trips $P = (p_1, p_2, \ldots, p_\ell)$ and of an ordered sequence of delivery trips $D = (d_1, d_2, \ldots, d_m)$ that must be synchronized: indeed the items are passed from the pickup area to the delivery area by means of a FIFO stack (a horizontal conveyor in the real application) whose input (resp. output) extremity is located at vertex 0 of D^1 (resp. D^2). Thus items collected by a pickup trip must be put on the stack before those collected by subsequent pickup trips. By the FIFO policy a delivery trip d_i preceding another delivery trip d_j must deliver items that, on the stack, precede those delivered by d_j .

For a trip t we indicate as $a \in t$ any arc between vertices which are consecutive in t. For every trip t and for $R = 1, 2$ let $c^R(t) = \sum_{a \in t} c^R(a)$ and for every sequence of pickup trips P let $c^1(P) = \sum_{i=1}^{\ell} c^R(p_i)$; symmetrically, $c^R(D) = \sum_{i=1}^{m} c^2(d_i)$. The objective of a problem in the SPDP-FS family is to find a solution (P, D) that minimizes $c^1(P) + c^2(D)$.

Example 1. In Fig. 1 we illustrate a small instance of a problem belonging to the SPDP-FS family. A solution is depicted in boldface: the pickup vehicle (left) performs first trip $(1, 5, 4)$ then trip $(2, 3)$. Hence items in $\{1, 4, 5\}$ must be delivered before items in $\{2, 3\}$. In the example they are in fact arranged in the order of visit. This has an impact on the delivery solution. In the most constrained setting, the delivery vehicle is forced to the inconvenient trip $(1, 5, 4)$ to respect such an order, as shown in Fig. 1. In fact, the order of visits is not the only degree of freedom of

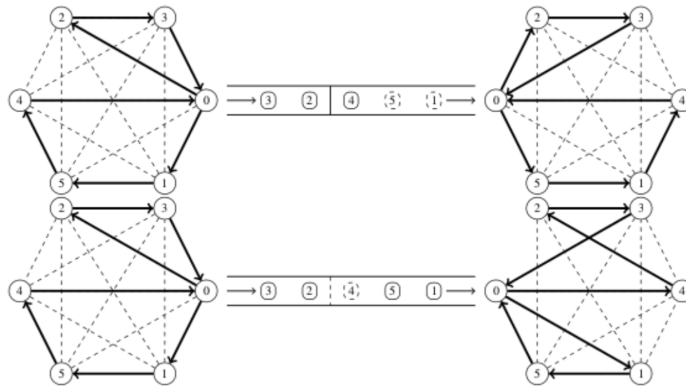

Fig. 2 Schematic illustration of the automated warehouse, when permutations are allowed (top) and when overlaps are allowed (bottom).

the vehicles, which may or may not be allowed to permute items on the stack after pickup or before delivery. This is what we call the *permutation* variant. For example, in Fig. 2 (top) the delivery vehicle is allowed to retrieve the items $\{1,4,5\}$ from the stack, changing their order from $(1,5,4)$ to $(5,1,4)$ thereby removing the need of a detour. Such an option may be symmetrically given to the pickup vehicle.

Besides permuting, delivery vehicles may or may not be allowed to mix items from subsequent pickup batches. This is what we call the *overlap* variant. For example, in Fig. 2 (bottom) the delivery vehicle is allowed to deliver $(1,5)$, wait for the next pickup batch to arrive, and finally deliver $(4,2,3)$, possibly improving delivery cost. Permutation and overlap may or may not be allowed simultaneously, giving more optimization potential at the expense of higher complexity in the real world process. For instance, the trip $(4,2,3)$ in the last example may be replaced by $(4,3,2)$, thus removing a detour. Pickup permutation and delivery permutation are not equivalent. For instance, a delivery trip like $(5,4,3)$ is feasible only due to pickup permutations, while a delivery trip like $(5,2,4)$ only due to delivery permutations.

The SPDP-FS variants. As previously mentioned, the synchronization between two sequences of pickup and delivery trips in a SPDP-FS solution may depend not only on the order in which items are collected, but also on how items are arranged on the stack. We therefore identify eight variants mixing the possibility of permuting items by each vehicle (or not) and the presence of constraints allowing (or disallowing) delivery trips to be overlapping different pickup trips. When permutation is allowed we are essentially modelling situations where the items can be loaded or unloaded without observing any particular order; when overlapping is disallowed, we are modelling situations where the stack must be empty before a new pickup trip can take place.

Formally, for every trip t let $V(t)$ be the set of vertices other than 0 visited by t. Let $P = (p_1, p_2, \ldots, p_\ell)$ be an ordered sequence of pickup trips such that the $V(p_i)$ form a partitioning of $V \setminus \{0\}$. P induces a partial order: if $v, w \in V \setminus \{0\}$, we write $v \prec_P w$ if $v \in p_i$ and $w \in p_j$ for some $1 \leq i < j \leq \ell$ and we write $v \not\prec_P w$ otherwise. A *P-sequence*, instead, is the (totally) ordered sequence of vertices in $V \setminus \{0\}$, in the order they are visited in P. Identical definitions hold for a partial order $\prec_D$ and a *D-sequence* w.r.t. a fixed ordered sequence of delivery trips $D = (d_1, d_2, \ldots, d_m)$. (P, D) form a feasible SPDP-FS solution if it satisfies:

$$|V(p_i)| \leq k_1 \qquad \forall i = 1, 2, \ldots, \ell \tag{1}$$

$$|V(d_j)| \leq k_2 \qquad \forall j = 1, 2, \ldots, m \tag{2}$$

$$V(p_1), V(p_2), \ldots, V(p_\ell) \text{ partition } V \setminus \{0\} \tag{3}$$

$$V(d_1), V(d_2), \ldots, V(d_m) \text{ partition } V \setminus \{0\} \tag{4}$$

Intuitively, SPDP-FS OVERLAP variants are those in which the FIFO stack is *not* required to be empty before a new pickup trip can take place. As a result, delivery trips can mix items from different pickup batches, as long as they are adjacent in the stack. Formally, the OVERLAP variants are defined as follows:

NO-PERMUTATION,OVERLAP. The pair (P, D) is a feasible solution if and only if it satisfies (1)–(4) and the P-sequence is identical to the D-sequence.

PERMUTATION,OVERLAP. The pair (P, D) is a feasible solution if and only if it satisfies (1)–(4) and for every $v, w \in V \setminus \{0\}$ such that $v \prec_P w$ it also holds $w \not\prec_D v$.

DELIVERY PERMUTATION,OVERLAP. The pair (P, D) is a feasible solution if and only if it satisfies (1)–(4) and:

- for every $j = 1, 2, \ldots, m$, $V(d_j)$ is a set of elements which are consecutive in the P-sequence;
- for every $v, w \in V \setminus \{0\}$, if $v \prec_D w$ then v precedes w in the P-sequence.

PICKUP PERMUTATION,OVERLAP. The pair (P, D) is a feasible solution if and only if it satisfies (1)–(4) and for every $v, w \in V \setminus \{0\}$ such that $v \prec_P w$ we also have that v precedes w in the D-sequence.

Symmetrically, in NO-OVERLAP variants the FIFO stack is required to be empty before new pickup trips, and therefore delivery trips do not mix items from different pickup trips. Their models are obtained from the four above by further adding the condition: $\forall j = 1, 2, \ldots, m,\ \exists$ a unique $i \in \{1, 2, \ldots, \ell\}$ s.t. $V(d_j) \subseteq V(p_i)$.

When the capacity values k_1 and k_2 are part of the input all SPDP-FS variants above are **NP**-hard. We reduce from the *Euclidean travelling salesman problem* (Euclidean-TSP) of which an instance is given by a complete graph $H = (W, E)$ with each vertex $w \in W$ corresponding to a point $\pi(w)$ in the Euclidean plane. Letting $c(v, w)$ be the Euclidean distance between $\pi(v)$ and $\pi(w)$, the Euclidean-TSP asks to find a minimum weight Hamiltonian tour on the weighted graph (H, c). Note that function c is *metric*, that is, it satisfies the *triangle inequality* $c(u, v) + c(v, w) \geq c(u, w)$. Nonetheless, the Euclidean-TSP is known to be **NP**-hard [6].

Proposition 1. *If k_1 and k_2 are part of the input, the SPDP-FS is **NP**-hard.*

Proof. Let (H,c) be a weighted complete graph on n vertices defining an Euclidean-TSP instance. We define a SPDP-FS instance on $D^1 = (H,c^1)$ and $D^2 = (H,c^2)$ with $k_1 = n$, $k_2 = 1$ and $c^1(u,v) = c(u,v)$ and $c^2(u,v) = 0$ for all u,v vertices of H. Since c^1 is metric and $k_1 = n$ a minimum cost Hamiltonian tour P of D^1 is also a minimum cost pickup sequence satisfying (1). Let the P-sequence be $(v_1,v_2,\ldots,v_n)$. The delivery sequence $D = ((v_1),(v_2),\ldots,(v_n))$ has cost 0; moreover (P,D) is feasible for all SPDP-FS variants. Hence the optimal value of the given SPDP-FS instance is the optimal value of the starting Euclidean-TSP instance.

Some variants of the SPDP-FS become solvable in polynomial time when the capacities k_1 and k_2 are very specific fixed values. The most immediate case is when $k_1 = k_2 = 1$ (independently on the variant): here all trips of a sequence visit exactly one vertex $v \in V \setminus \{0\}$. Then, all pairs of sets $\mathscr{P} = \{p_1, p_2, \ldots, p_\ell\}$ of pickup trips and $\mathscr{D} = \{d_1, d_2, \ldots, d_m\}$ of delivery trips, when an arbitrary (but identical) ordering is applied to both, yield a feasible pair (P,D) of pickup and delivery sequences for every variant. Moreover, the cost of such a solution does not depend on the chosen order.

The cases with larger fixed values of k_1 and k_2 are more involved. In fact, we are able to provide a polynomial-time algorithm only for the NO-OVERLAP variants with $k_1,k_2 \in \{1,2\}$.

Proposition 2. *If $k_1,k_2 \in \{1,2\}$ there exists a polynomial-time algorithm solving the* NO-OVERLAP *variants of the SPDP-FS.*

Proof. We start by calculating for all $v \in V \setminus \{0\}$ the total cost $c[v]$ of visiting only v in a pickup trip and a delivery trip, *i.e.*, $c[v] = c^1((v)) + c^2((v))$. We also compute, for every $v,w \in V \setminus \{0\}$, the minimum total cost $c[v,w]$ of visiting v and w in the same pickup trip and of delivering them in any manner by satisfying the NO-OVERLAP variant under consideration. Let $t[v,w]$ denote the sequences of pickup and delivery trips attaining value $c[v,w]$. The whole pre-processing phase takes $O(n^2)$ time.

Now, for every $v \in V \setminus \{0\}$ we let v' and v'' be two copies of v. We consider H' and H'' two complete graphs on vertex sets $W' = \{v' : v \in V \setminus \{0\}\}$ and $W'' = \{v'' : v \in V \setminus \{0\}\}$ respectively. From the union of H' and H'' we create a new graph H obtained by further linking vertices v' and v'' for every $v \in V \setminus \{0\}$. Finally, we associate weights s to the edges of H with $s_e = c[v,w]$ if $e = \{v',w'\}$ for some $v,w \in V \setminus \{0\}$, $s_e = c[v]$ if $e = \{v',v''\}$ for some $v \in V \setminus \{0\}$ and $s_e = 0$ otherwise. A perfect matching of minimum weight in (H,s) corresponds to an optimal solution for the considered NO-OVERLAP variant: just perform the trips in $t[v,w]$ and trip (u) for every $u,v,w \in V \setminus \{0\}$ such that $\{v',w'\}$ and $\{u',u''\}$ respectively belong to such a perfect matching. The proposition holds since a perfect matching of minimum weight can be computed in polynomial time on every graph [4].

The algorithm provided in the proof of Proposition 2 easily generalizes to all SPDP-FS variants if $k_1,k_2 \in \{1,2\}$ and at least one capacity value is 1. However, when $k_1 = k_2 = 2$, the same algorithm does not work outside NO-OVERLAP variants.

The last observation suggests that NO-OVERLAP variants may have some structural feature which is not in common with the others. Supporting this intuition, it is not hard to prove that (P,D) is a feasible solution to a PERMUTATION,NO-OVERLAP problem if and only if it is a feasible solution to the corresponding PICKUP PERMUTATION,NO-OVERLAP problem. This immediately implies that if (P,D) is a feasible solution to the DELIVERY PERMUTATION,NO-OVERLAP problem then it is a solution to the corresponding PICKUP PERMUTATION,NO-OVERLAP problem. Such a phenomenon does not occur on OVERLAP variants.

In fact, we have analysed the inclusion relations between the sets of solutions of the eight variants building a full set-inclusion hierarchy. To keep the focus of the paper we omit these results. However, our main conclusion is that NO-PERMUTATION variants take a relevant place in such a hierarchy, and are therefore good candidates to start a structural investigation on the whole SPDP-FS family.

3 Algorithms for Sub-Problems of NO-PERMUTATION Variants

From now on we restrict to NO-PERMUTATION variants; in particular, we consider relevant subproblems arising when only part of a solution is given, and either feasibility must be checked or optimal completion must be found.

NO-PERMUTATION feasibility. Let us assume to have a set $\mathscr{P} = \{p_1, p_2, \ldots, p_\ell\}$ of pickup trips and a set $\mathscr{D} = \{d_1, d_2, \ldots, d_m\}$ of delivery trips such that (1) and (2) hold. The *feasibility problem* is to independently find an ordering of the elements of $\mathscr{P}$ and $\mathscr{D}$ so that the resulting sequences P and D represent a feasible solution, or to prove that no such ordering exists.

We start with the NO-PERMUTATION,OVERLAP case. Let us denote as *starting* each vertex which is the first vertex of two trips in $\mathscr{P} \cup \mathscr{D}$. A trip t' is said to be *contained* in t if it is a subsequence of t. Two trips $t = (u_1, u_2, \ldots, u_k, v_1, v_2, \ldots, v_\ell)$ and $t' = (v_1, v_2, \ldots, v_\ell, w_1, w_2, \ldots, w_m)$ are said *overlapping*: they respectively end and start by a same non-empty subsequence; note that there can be at most one trip overlapping with a given trip t. We can solve the feasibility problem for $\mathscr{P}$ and $\mathscr{D}$ in polynomial time. We omit the details of the algorithm, but its general idea is to consider a set $T = \mathscr{P} \cup \mathscr{D}$, and then to pick a starting vertex v and one of the trips t containing it, removing both t and all trips t' contained in t from T, to either update t with a possible trip overlapping with t or pick another starting vertex and to iterate, until no update is possible. The problem is feasible if and only if $T = \emptyset$ at the end of this procedure.

In the NO-PERMUTATION,NO-OVERLAP case we first check that every pair of trips $p \in \mathscr{P}$ and $d \in \mathscr{D}$ such that $V(p) \cap V(d) \neq \emptyset$ also satisfies $V(d) \subseteq V(p)$. If at least one pair violates the condition then $\mathscr{P}$ and $\mathscr{D}$ are infeasible. Otherwise we run the iterative algorithm for the NO-PERMUTATION,OVERLAP case.

NO-PERMUTATION splitting. Now we turn our attention to the following sub-problem: a stack configuration is given as input (together with data), and the task is

to find pickup and delivery sequences of minimum cost which are *consistent with the given configuration*. In other words, we assume that the pickup and delivery sequences are fixed, while the depot return operations forming trips remain to be optimized. Let $S = (v_1, \dots, v_n)$ be a sequence of elements in $V \setminus \{0\}$. It represents the order of the corresponding items on the stack. A *splitting* of S is a partition of S into subsequences that respect its order. For example, if $S = (1,2,3,4,5)$ one of its splitting is $((1,2),(3,4,5))$, while $((3,4,5),(1,2))$ is not because it does not respect the order of S. Here we consider both NO-PERMUTATION variants, thus the trips of P and D in a feasible SPDP-FS solution (P,D) are splittings of $(1,2,\dots,n)$. The *optimal sequence splitting problem* is to find two splittings P and D of $(1,2,\dots,n)$, minimizing $c^1(P) + c^2(D)$ and such that (P,D) is feasible for the considered NO-PERMUTATION variant. Up to renaming the vertices, we give polynomial-time algorithms to solve this problem for $S = (1,2,\dots,n)$.

For what concerns NO-PERMUTATION,NO-OVERLAP, we use an acyclic network $\mathscr{N}$ whose vertices are labelled (i,j) with $0 \le j \le i \le n$. For every $i = 0,1,\dots,n$ there are arcs from vertex (i,i) to all vertices (k,i) with $i < k \le \min\{i+k_1,n\}$. For every $i = 1,2,\dots,n$ and $j = 1,2,\dots,i-1$ there are arcs from vertex (i,j) to all vertices (i,k) with $j < k \le \min\{j+k_2,i\}$. Vertex (i,j) is interpreted as a state in the construction of the optimal pickup and delivery splittings P and D, namely it indicates that all vertices from 1 to i have been visited by some pickup trip and all vertices from 1 to j have been visited by some delivery trip. Thus, the arc from (i,i) to $(i+k,i)$ for some $1 \le k \le k_1$ corresponds to extending the current state (i,i) into state $(i+k,i)$ by performing $p = (i+1,i+2\dots,i+k)$ as next pickup trip. Similarly, the arc from (i,j) to $(i,j+k)$ corresponds to performing $d = (j+1,j+2,\dots,j+k)$ as next trip in the optimal delivery sequence D. By construction, $|V(p)| \le k_1$, $|V(d)| \le k_2$ for all pickup and delivery trips p and d as above. Hence a path from $(0,0)$ to (n,n) corresponds to a pair (P,D) of pickup and delivery splittings of S; moreover, the P-sequence and D-sequence of these splittings coincide. Finally, representing $\mathscr{N}$ as in Fig. 3 (left) the arcs traversing $\mathscr{N}$ "vertically" only leave from vertices on the "diagonal". This property guarantees that each trip in the delivery splitting is contained in a trip of the pickup splitting. To see this, let us fix a path in $\mathscr{N}$. This gives a splitting of S into pickup and delivery trips. Let $d = (j_1,\dots,j_2)$ be one such a delivery trip. It corresponds to the arc of the path linking (i,j_1) to (i,j_2) for some $i \ge j_2$. Letting j_0 be the smallest index such that (i,j_0) is traversed by the chosen path, the vertex preceding (i,j_0) in the path is necessarily (j_0,j_0). It follows that the pickup splitting of S contains the trip $p = (j_0,\dots,i)$. Since $j_0 \le j_1$ and $i \ge j_2$ we get $V(d) \subseteq V(p)$. It follows that every path in $\mathscr{N}$ from $(0,0)$ to (n,n) corresponds to a splitting of S which is also a NO-PERMUTATION,NO-OVERLAP solution. In fact the latter is a one-to-one correspondence as it can be easily verified by an analogous reasoning. An example is given in Fig. 3 (left).

In a pre-processing step we calculate, in polynomial time, the cost $c^1[i_1,i_2]$ of the trip corresponding to each arc between (i_1,j) and (i_2,j) as well as the cost $c^2[j_1,j_2]$ of the trip corresponding to each arc between (i,j_1) and (i,j_2). Associating these costs to the corresponding arcs in $\mathscr{N}$ we obtain an acyclic digraph with nonnegative weights. A path of minimum weight from $(0,0)$ to (n,n) gives an optimal splitting of

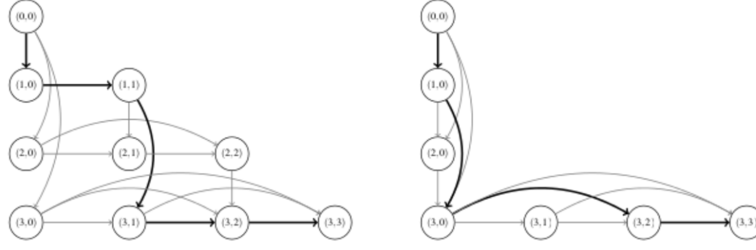

Fig. 3 Networks $\mathcal{N}$ for the optimal sequence splitting problem with $n = k_1 = k_2 = 3$ in the cases NO-PERMUTATION,NO-OVERLAP (left) and NO-PERMUTATION,OVERLAP (right). The boldface path in the left network corresponds to the splittings $P = ((1),(2,3))$ and $D = ((1),(2),(3))$. The boldface path in the right network corresponds to the splitting $P = ((1),(2,3))$ and $D = ((1,2),(3))$.

S by the arc-trip correspondence explained above. Being $\mathcal{N}$ acyclic, very efficient algorithms can be used to compute such a path [5, Sect. 24.2].

For what concerns NO-PERMUTATION,OVERLAP, instead, we use a network $\mathcal{N}$ whose vertices are labelled $(i,0)$ and (n,j) for $0 \leq i,j \leq n$. For every $i = 0,1,\ldots,n$ there is an arc from vertex $(i,0)$ to all vertices $(k,0)$ with $i < k \leq \min\{i+k_1,n\}$; similarly, for every $j = 0,1,\ldots,n$ there is an arc from (n,j) to all vertices (n,k) with $j < k \leq \min\{j+k_2,n\}$. An example is given in Fig. 3 (right). The arc-trip correspondence is the same as in the NO-PERMUTATION,NO-OVERLAP case. Associating costs to the arcs as we did for that variant, we compute an optimal pair of pickup and delivery splittings by finding a minimum cost shortest path in $\mathcal{N}$.

4 Computational Results

Finally, we carried out experiments to assess the practical applicability of the optimal splitting algorithms presented in Section 3 for both NO-PERMUTATION variants.

Algorithms. We designed dynamic programming algorithms which are able to compute an optimal sequence splitting when a stack configuration is given.

Their structure is simple: once the appropriate network $\mathcal{N}$ is built, we consider the vertices in the topological order induced by their labels, that is, row-by-row and, for each row, column-by-column. Then, we assign *partial splitting costs* to each vertex, with a procedure similar to [5, Sect. 24.2]. The partial splitting cost of the starting vertex $(0,0)$ is initialized to 0. When a vertex is considered, its partial splitting cost is set by looking at all its incoming arcs, and computing the minimum among the cost of each arc plus the partial splitting cost of the corresponding starting vertex. At the end of this procedure, the partial splitting cost of the vertex (n,n) corresponds to an optimal (complete) one. A corresponding solution can be found by keeping track of the arcs defining the minima.

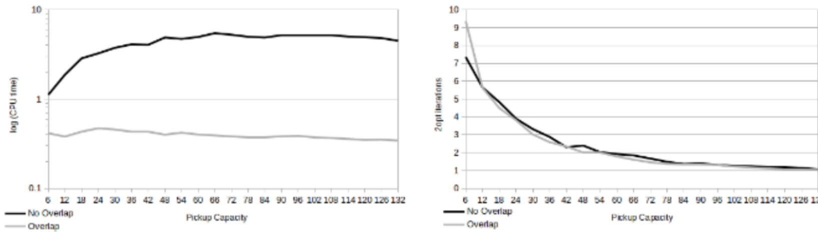

Fig. 4 Performance of heuristics.

For testing, we took as starting stack configuration the sequence of items corresponding to an optimal Hamiltonian tour of a complete graph having one vertex for each item, and one arc (i, j) between each pair of items i and j, whose cost is the sum of pickup and delivery distances between i and j. Clearly, an optimal splitting of such a Hamiltonian tour is not guaranteed to produce and optimal SPDP-FS solution. Therefore we have also experimented with a simple 2-opt local search mechanism: we generate the full 2-opt neighborhood of the current solution and score it by solving an optimal splitting problem for each of its elements. We take the best solution in the neighborhood and we iterate, until a local minimum is reached.

As a trivial bound, we computed the optimal Hamiltonian tours of pickup and delivery item graphs independently, and summed up their values: this is clearly a relaxation of the problem, neglecting both the vehicle capacities and the effect of the linking stack. Our algorithms have been implemented in C++ and compiled with gcc version 7.2 with full optimization options. Optimal Hamiltonian tours have been computed by means of the Concorde library [3]. Tests were run on a linux PC equipped with an i7-3630QM CPU running at 2.40GHz, single thread.

Datasets. We considered a dataset of pickup and delivery instances originally designed for the double travelling salesman problem with multiple stacks [7]. Our dataset includes three classes of instances, respectively containing 33, 66 and 132 *pairs* of pickup and delivery vertices. Ten instances are given for each class. Distances were set as the euclidean ones computed from the input coordinates, rounded to the nearest integer and finally corrected with an all-pair shortest path procedure to ensure triangle inequalities to be respected. For each instance having 33 (resp. 66) pairs of vertices, we considered pickup vehicle capacities from 3 to 33 (resp. 66) in steps of 3, and delivery vehicle capacities from 3 to the pickup capacity value in steps of 3. For each instance having 132 pairs of vertices we considered instead capacities from 6 to 132 in steps of 6, and delivery vehicle capacities from 6 to the pickup capacity value in steps of 6. This procedure yielded a dataset of 11440 instances for each variant.

Computing time. The computing time for solving a single optimal splitting problem was always negligible (below $0.01s$ for both variants, independently on the instance size and capacity). The computing time for the full 2-opt local search process was also negligible on instances with either 33 or 66 pairs of vertices, independently

variant	size	Avg. overhead	Avg. impr.	Avg. opt.
NO-OVERLAP	33	10.05%	0.84%	47.14%
	66	8.17%	0.61%	54.27%
	132	5.10%	0.41%	59.65%
OVERLAP	33	6.50%	0.76%	45.09%
	66	5.56%	0.52%	53.00%
	132	3.66%	0.33%	59.07%

Table 1 Quality of heuristics and bounds.

from the pickup and delivery capacity values. Therefore, in Figure 4 (left) we report the computing time (y axis) for the full 2-opt process only on instances with 132 pairs of vertices (using logarithmic scale, on both variants), averaged by pickup capacity value (x axis); in Figure 4 (right) we report instead the number of 2-opt moves required to reach a local minimum (y axis), on the same instances and performing the same aggregation. Values related to the OVERLAP variant are depicted in grey, those related to NO-OVERLAP in black.

As expected, higher capacity values yield higher computing times: the optimal splitting graphs are more dense, and more checks are needed in the dynamic programming algorithms. The number of 2-opt moves required to reach a local minimum decreases as the capacity increases. We argue that higher capacity values yield less overhead costs for the vehicle to return to the stack; as a consequence, being the routing part of the cost dominant, the Hamiltonian tour is already similar to a local minimum. The increase of overall CPU time is not monotone: for very high pickup capacity values the reduced number of 2-opt moves balances the higher CPU time needed during a single iteration.

Solutions quality. We denote as IS the value of the initial solution, as DA the sum of costs of arcs leaving and returning to the depot for intermediate stops in such a solution, as OS the solution value after 2-opt, and TB the value of the trivial bound. In Table 1 we report in turn, for each class of instances, for both OVERLAP and NO-OVERLAP variants, the average fraction of the initial solution cost for the vehicle to return to the stack with intermediate stops (overhead, DA/IS), the average overall improvement yielded by 2-opt (impr., (TO-IS)/TO) and the average gap between the value of the solution found by 2-opt and the trivial bound (opt., (TO-TB)/TO).

The results show that the overhead decreases as the number of items increases; this is most probably a statistical side effect: when the set of items is large, and the depot is in the baricenter of the item locations (as in our instances) the probability of passing near the depot in a random connection is higher. The average improvement yielded by 2-opt is always very low, and decreasing as the number of items increases. At the same time, the gap between local minima solution values and the trivial bound is very large, and increases as the number of items increases. The values in these three columns make us conjecture that by computing an optimal Hamiltonian tour, and then optimally split it, good heuristic solutions can be achieved. However, we suspect the trivial bound to be very poor, thereby asking for better lower bounding procedures.

5 Conclusions

From a modelling point of view, we restricted the complexity of SPDP-FS from industry by considering two peculiar features: the possibility of changing the order of items during loading or unloading operations to and from the conveyor, and the possibility of starting pickup trips even if the conveyor is not empty.

We have shown that all variants arising from the combination of these features are **NP**-hard, although some of them admit polynomial-time resolution algorithms when the capacities of the vehicles are fixed to small values *i.e.,* 1 or 2.

By focusing on the NO-PERMUTATION variants, we could find polynomial-time algorithms for two relevant sub-problems: checking feasibility when the set of pickup and delivery trips are given (but the order of items in the conveyor is unknown), and optimizing the pickup and delivery trips when the order of items in the conveyor is given.

Our algorithms proved to be effective also from an experimental point of view: tests on the double travelling salesman problem with multiple stacks showed that a single sub-problem resolution can be carried out in fractions of a second; their embedding in a simple local search algorithm provided promising results.

Since, on the contrary, trivial lower bounds appear very weak, our current research is focused on the design of good quality ones.

Acknowledgements This research was partially funded by Regione Lombardia, grant agreement n. E97F17000000009, project AD-COM, and Università degli Studi di Milano, Dipartimento di Informatica, Piano Sostegno alla Ricerca 2016-2020.

References

1. AD-COM: ADvanced Cosmetic Manufacturing (2020). https://ad-com.net/
2. Applegate, D.L., Bixby, R.E., Chvatal, V., Cook, W.J.: The Traveling Salesman Problem: A Computational Study. Princeton University Press (2006)
3. D. L. Applegate, R. E. Bixby, V. Chvatal and W. J. Cook: Concorde (2003). http://www.math.uwaterloo.ca/tsp/concorde.html
4. Edmonds, J.: Maximum matching and a polyhedron with 0, 1-vertices. Journal of research of the National Bureau of Standards B **69**(125-130), 55–56 (1965)
5. Leiserson, C.E., Rivest, R.L., Cormen, T.H., Stein, C.: Introduction to Algorithms, vol. 6. MIT press Cambridge, MA (2001)
6. Papadimitriou, C.H.: The Euclidean travelling salesman problem is NP-complete. Theoretical computer science **4**(3), 237–244 (1977)
7. Petersen, H.L., Madsen, O.B.: The double travelling salesman problem with multiple stacks–formulation and heuristic solution approaches. European Journal of Operational Research **198**(1), 139–147 (2009)
8. Toth, P., Vigo, D.: The vehicle routing problem, 2nd edition. SIAM (2014)