European Journal of Operational Research 249 (2016) 1074–1081

Contents lists available at ScienceDirect

European Journal of Operational Research

journal homepage: www.elsevier.com/locate/ejor

Production, Manufacturing and Logistics

Complexity results for storage loading problems with stacking constraints

Florian Bruns[a], Sigrid Knust[a,*], Natalia V. Shakhlevich[b]

[a] University of Osnabrück, Institute of Computer Science, Osnabrück 49069, Germany
[b] School of Computing, University of Leeds, Leeds LS2 9JT, UK

ARTICLE INFO

Article history:
Received 24 June 2014
Accepted 21 September 2015
Available online 30 September 2015

Keywords:
Storage loading
Stacking
Complexity
Stacking constraints

ABSTRACT

In this paper, we present complexity results for storage loading problems where the storage area is organized in fixed stacks with a limited common height. Such problems appear in several practical applications, e.g., in the context of container terminals, container ships or warehouses. Incoming items arriving at a storage area have to be assigned to stacks so that certain constraints are respected (e.g., not every item may be stacked on top of every other item). We study structural properties of the general model and special cases where at most two or three items can be stored in each stack. Besides providing polynomial time algorithms for some of these problems, we establish the boundary to NP-hardness.

1. Introduction

Storage loading problems arise in several practical applications, e.g., in the context of container terminals, container ships, warehouses or steel yards. In such problems incoming items arrive at a storage area by trains, vessels or trucks and have to be assigned to stacks respecting certain constraints. Usually, only the topmost item of each stack can be directly retrieved, i.e., the items are accessed in last-in-first-out order. This implies that if an item stacked below has to be retrieved, a so-called reshuffling (relocation) operation is necessary. In this paper, we only consider the loading process and assume that no outgoing items have to be retrieved during this process.

We study problems where items have to be loaded into a two-dimensional storage area consisting of stacks where each stack has its own fixed position. This means that one cannot decide where to position a stack in the area, but which stack to choose for placing an item. Such a predefined arrangement of stacks is motivated by yard areas in maritime or rail-road terminals, which are often organized in a “fixed grid” layout. These layouts have fixed subareas with predefined lengths, and containers must be placed into these subareas without exceeding their borders. If we assume that only one stack may be opened in each subarea, this case is equivalent to the situation with fixed positions of stacks.

The items are usually relocated by cranes moving above the stacks, which imposes a restriction on the maximum height of a stack. We assume that each stack cannot hold more than b items. In some scenarios the items may have additional characteristics like weights or heights. Then, in addition to the number b that limits the number of items in any stack, one has a height limit or a weight limit for each stack. Also, special restrictions on the containers’ locations may exist. For example, reefer containers need a power socket, so only locations with an appropriate configuration are feasible for them.

The main goal of a storage loading problem is to assign each incoming item to a feasible position in a stack such that a given objective function is optimized. Several aspects of such problems are of interest for practitioners. For example, the distances the cranes move should be minimized to reduce the energy costs for operating the cranes. Another objective is to achieve an allocation of containers so that future reshuffles related to storage unloading are avoided because they are time- and energy-consuming. Since in general the exact number of required reshuffles cannot be easily determined in advance, this objective is often replaced by a lower bound on the number of reshuffles, which is easier to compute. For this purpose, the number of “unordered stackings” may be minimized, which is defined as the total number of vertically adjacent items ordered in the wrong way (e.g., with respect to retrieval times). Another objective important in practice is to minimize the total number of items stacked on levels above the ground level (this reduces the risk of reshuffling). In contrast to that, some practitioners prefer to use as few stacks as possible to have more flexibility for the remaining stacks in the storage area.

As observed by Lehnfeld and Knust (2014), up to now almost no complexity results for storage loading problems have been published. On the other hand, for storage unloading problems or combined loading/unloading problems some results are known (cf.

* Corresponding author. Tel.: +49 541 969 2483.
E-mail addresses: florian.bruns@uni-osnabrueck.de (F. Bruns), sigrid.knust@uni-osnabrueck.de (S. Knust), N.Shakhlevich@leeds.ac.uk (N.V. Shakhlevich).

http://dx.doi.org/10.1016/j.ejor.2015.09.036

Lehnfeld & Knust, 2014). For example, Caserta, Schwarze, and Voß (2012) deal with a basic unloading problem (the "blocks relocation problem") where items are stored in stacks and have to be retrieved in a given order. It is shown that minimizing the number of reshuffles is NP-hard. In the so-called "container stowage problem", a vessel visits several ports consecutively. At each port, a set of containers has to be retrieved from the vessel and another set of containers has to be stored on it. Avriel, Penn, and Shpirer (2000) derived complexity results by relating this problem to the coloring of special graphs: while for the problem with unlimited b and at most 3 stacks it can be decided in polynomial time whether there exists a solution without relocations, for every fixed number of stacks greater than 3 the problem becomes NP-complete. Delgado, Jensen, Janstrup, Høyer Rose, and Høj Andersen (2012) consider a loading problem where a container ship has to be loaded with a set of items. According to the vessel's stability, weight and height restrictions are given for each stack. Since some containers need a power socket, there are location restrictions as well. By a reduction from the bin packing problem, it is shown that minimizing the number of used stacks is NP-hard.

Kim, Park, and Ryu (2000) deal with a loading problem where a sequence of items has to be stored in a storage yard. It is assumed that each item belongs to one of three weight groups, but the weight group of each item is not known before its arrival. The objective is to minimize the expected number of reshuffles. A dynamic programming approach is described based on the probability of the weight group of the next arriving item. Kang, Ryu, and Kim (2006) tackle a similar problem by simulated annealing. In the context of rail-road terminals, Jaehn (2013) considers a storage loading problem where containers from a bundle of trains arriving simultaneously have to be placed into a storage area consisting of parallel lanes. After showing NP-hardness of two models, heuristic algorithms are presented and tested on real-world data. An overview of optimization approaches in rail-road terminals has recently been provided by Boysen, Fliedner, Jaehn, and Pesch (2013). Storage loading problems in seaport container terminals are considered in Borgmann, van Asperen, and Dekker (2010) and Dekker, Voogd, and van Asperen (2006). Departure times are given and the aim is to avoid stacks that cause reshuffles when an item is stacked on top of another item with an earlier departure time.

The main objective of our paper is to provide first theoretical results for storage loading problems with stacking constraints. In addition to NP-hardness results, we derive polynomial time algorithms for special situations which may be used as building blocks in heuristic approaches for more general problems with additional constraints typical for real-world applications.

The remainder of the paper is organized as follows. In Section 2, we formally introduce typical constraints and objectives that occur in practical storage loading problems. In Section 3, we consider special cases with the stacking limit $b = 2$. We show that minimizing the number of used stacks, minimizing the number of items stacked above the ground level and minimizing the number of unordered stackings, can be done in polynomial time even if there are compatibility constraints for pairs of items. In Section 4, we prove that for the stacking limit $b = 3$ in the presence of stacking constraints it is already strongly NP-complete to decide whether a feasible solution exists or not. Afterwards, in Section 5, we consider the more general situation of an arbitrary stacking limit b and identify two special cases that are polynomially solvable. Finally, conclusions are presented in Section 6.

2. Problem formulation

In this section, we give a formal definition of storage loading problems and introduce notations for its various versions. As mentioned before, we focus on problems where the storage area is arranged in stacks and the positions of the stacks in a two-dimensional area are given. It is assumed that the yard layout is fixed and the stack positions cannot be changed.

Let m be the number of stacks where for each stack a position in the yard is specified by x- and y-coordinates. Each stack can hold at most b items. The set of all items is denoted by $I = \{1, 2, \dots, n\}$ where normally the inequality $m < n$ holds, i.e., some items have to be stacked on others. We assume that $n \le bm$; otherwise the problem is infeasible.

For each item $i \in I$ its original position O_i is given by its x- and y-coordinates. Typically, positions O_i correspond to locations of items on an arriving train or truck. Additionally, for a non-empty storage area the locations of the items already stored in the stacks are given, specified by a stack number and a level in the stack. We assume that these positions are fixed and cannot be changed. We denote by $I^{fix} \subset I$ the set of fixed items in the storage area and by $I^{in} \subseteq I$ the set of incoming items. Since the items in I^{fix} are not allowed to be re-allocated, we exclude from consideration all stacks which are already completely filled with items and assume that in each of the remaining m stacks at least one free position exists; the corresponding items are removed from the sets I^{fix} and I.

Usually, in practice there are restrictions concerning feasible stacking configurations. For example, heavier items are not allowed to be stacked on top of lighter ones, longer items are not allowed to be put on shorter ones, and items of different material or with different destinations may not be put on each other. All such stacking constraints may be encoded by a 2-dimensional binary matrix $S = (s_{ij})_{n\times n}$, where $s_{ij} = 1$ if i can be stacked onto j and $s_{ij} = 0$ otherwise. These constraints can also be represented by a directed graph with n vertices and arcs $i \to j$ if $s_{ij} = 1$. Stacking constraints may be transitive (i.e., if item i is stackable on top of item j and item j is stackable on top of item h , then also i is stackable on top of h) or may have an arbitrary structure. For example, restrictions coming from weights or lengths of the items are transitive, while restrictions induced by materials may be non-transitive. Weight or length restrictions have the special property that all items are comparable (i.e., for all $i \ne j$ we have $s_{ij} = 1$ or $s_{ji} = 1$). Thus, these restrictions define a total order on all items. Sometimes for the items $i \in I$ departure (retrieval) times d_i are known indicating the time at which item i is expected to leave the storage area. In this situation, it is advantageous to stack an item i on top of j only if $d_i \le d_j$ holds; otherwise, a reshuffle will be necessary in the future.

Items stored in a stack are defined by a tuple $(i_k, \dots, i_1)$, where i_λ denotes the item stacked at level λ and $\lambda = 1$ corresponds to the ground level. Such a tuple is feasible if $k \le b$ and $s_{i_{\lambda+1}, i_\lambda} = 1$ for all $\lambda = 1, \dots, k-1$.

The simplest version of a storage loading problem is the feasibility problem which asks whether all items can be feasibly allocated to the storage area respecting all constraints, e.g., the stack capacity b, the stacking constraints S, a weight limit per stack or location restrictions. If this is possible, the objective is to assign each item to a feasible location (specified by a stack number and a level in the corresponding stack) minimizing one of the following objective functions:

- the total number $\#St$ of used stacks;
- the total number $\#SI^{>1}$ of items stacked at levels above the ground level (to reduce the risk of reshuffling);
- the total number of "unordered stackings" $\#US$ with respect to departure times d_i (to minimize the number of reshuffles): we count all pairs of items (i, j) where a less urgent item i is stacked directly on top of a more urgent item j, i.e., one with $d_j < d_i$;
- the total transportation costs TC for moving items from their original positions O_i to the assigned locations in the storage area. In general these costs depend on the x- and y-coordinates in the storage area as well as the assigned level (z-direction). Special situations are $TC(x, y)$ (the transportation costs only depend on the

x- and y-coordinates, the assigned level is negligible) or $TC(x)$ (the transportation costs only depend on the x-coordinates).

Note that the introduced functions can be considered independently or in a combined way. For example, $w_1 TC + w_2 \#SI^{>1}$ represents a weighted sum of transportation costs and the number of items stacked above the ground level with associated weights $w_1, w_2 \geq 0$.

We use the three-field notation $\alpha|\beta|\gamma$, introduced in Lehnfeld and Knust (2014), to represent the described versions of the storage loading problem. The problem type (e.g., loading or unloading) is specified in the first field α. In this paper, $\alpha = L$ for loading. Furthermore, we write $b = b'$ if the stacking limit b is fixed to $b' \in \mathbb{N}$ (e.g., $b = 2$ or $b = 3$).

The second field specifies the incoming items and characterizes additional storage restrictions. In loading problems, there is usually a set of items, denoted by I^{in}, which arrive simultaneously by a single delivery vehicle (e.g., by a single train, ship or truck). Sometimes a sequence π^{in} is given which means that the items have to be placed into the storage area according to a fixed sequence (e.g., if all items arrive on different trucks or a train has to be unloaded from "left to right" according to the original positions O_i). More generally, $(I^{in})_K$ denotes a sequence of K incoming sets, e.g., if several trains arrive consecutively and have to be unloaded in the order of their arrival. In this case, all items belonging to the same set have to be loaded into the storage area before any item of the next set can be stored. By default, we assume that there may be fixed items I^{fix} in the storage area and do not include I^{fix} in the notation. Otherwise, we explicitly write $I^{fix} = \emptyset$ in the β-field. Additional conditions in this field may include the entry s_{ij} to denote stacking constraints, and the entries "*weight-limit*" or "*height-limit*" indicating weight and height restrictions per stack, respectively.

Finally, the objective function is specified in the third field γ. In the feasibility version of a problem, no objective function is given (indicated by "–"), and the task is to find a feasible assignment of items to locations. For example, $L, b = 3 \mid I^{in}, s_{ij} \mid -$ denotes the feasibility problem with stacking limit $b = 3$ and arbitrary stacking constraints s_{ij}, while $L|I^{in}|TC(x, y)$ denotes the problem of assigning items to stacks with limit b (an arbitrary value given as part of the input) without stacking constraints minimizing transportation costs in x- and y-directions.

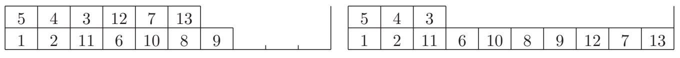
3. Problems with stacking limit *b* = 2

In this section, we consider problems where the stacking limit is $b = 2$. Such a situation is typical for rail-road container terminals, where often at most 2 containers may be stacked. We consider arbitrary stacking constraints s_{ij} and the objective functions $\#St$, $\#SI^{>1}$, and $\#US$. The storage area may contain some fixed items I^{fix} or may be empty.

Theorem 1. *Problem $L, b = 2 \mid I^{in}, s_{ij} \mid \#St$ with arbitrary stacking constraints s_{ij} can be solved as a maximum cardinality matching problem in $\mathcal{O}(n^{2.5})$ time. A solution minimizing $\#SI^{>1}$ can be derived from a feasible solution to $L, b = 2 \mid I^{in}, s_{ij} \mid \#St$ in $\mathcal{O}(n)$ time.*

Proof. We introduce the undirected graph $G = (V, E)$, in which the nodes V correspond to the items I and edges $E = \{\{i, j\} \mid i, j \in V\}$ connect two nodes if the corresponding items can be stored together in a stack. For incoming items $i, j \in I^{in}$ an edge $\{i, j\}$ exists if i can be stacked on j or vice versa, i.e., if $s_{ij} + s_{ji} \geq 1$. For items $i \in I^{in}$ and $j \in I^{fix}$ an edge $\{i, j\}$ exists if i is stackable on top of j, i.e., $s_{ij} = 1$.

By calculating a matching of maximum cardinality in the graph G, we get a solution with the largest number of stacks containing two items. Additionally, the items that are not matched have to be stored at the ground level. Thus, the total number of used stacks is minimized. A feasible solution with at most m stacks exists if and only if the number of edges in the matching plus the number of unmatched nodes is not larger than m. Since the number of nodes is n, a maximum cardinality matching can be computed in $\mathcal{O}(n^{2.5})$ time (cf. Even & Kariv, 1975).



(a) A feasible solution (b) An optimal solution minimizing $\#SI^{>1}$

Fig. 1. Constructing an optimal solution from a feasible one (Example 1).

Problem $L, b = 2 \mid I^{in}, s_{ij} \mid \#SI^{>1}$ minimizing the number of items stacked above the ground level can be solved as follows. We start with a solution to $L, b = 2 \mid I^{in}, s_{ij} \mid \#St$ and re-allocate as many items as possible to the ground level until all stacks contain at least one item. $\square$

Example 1. In the following, we illustrate how a solution minimizing $\#SI^{>1}$ can be constructed from a solution to problem $L, b = 2 \mid I^{in}, s_{ij} \mid \#St$. We consider an example with $m = 10$ stacks and $n = 13$ items, numbered from 1 to 13. We assume that in the maximum cardinality matching the edges {1, 5}, {2, 4}, {3, 11}, {6, 12}, {7, 10}, {8, 13} are chosen. Furthermore, item 9 is not matched at all.

The solution to the matching problem can be interpreted as the stacking solution shown in Fig. 1a. In this solution, three stacks are left empty, so three stacked items can be moved to the ground level. This results, for example, in the solution shown in Fig. 1b, where the number of items stacked above the ground level is $3 = 13 - 10$. Obviously, for $m = 10$, $b = 2$ and $n = 13$ this is optimal.

Next, we consider the same problem setting but with the objective to minimize the number of unordered stackings. For this problem additionally departure times d_i are given for all items $i \in I$.

Theorem 2. *Problem $L, b = 2 \mid I^{in}, s_{ij} \mid \#US$ with arbitrary stacking constraints s_{ij} can be solved as a minimum-weight perfect matching problem in $\mathcal{O}(n^3)$ time.*

Proof. At first we introduce $2m - n$ dummy items, leading to $2m$ items in total. These dummy items are used to represent empty positions in a solution and can be stacked together with any other item (real or dummy), in both directions.

We introduce the undirected graph $G = (V_1 \cup V_2, E_1 \cup E_2)$, where V_1 and V_2 represent real and dummy items, respectively, $|V_1| = n$, $|V_2| = 2m - n$. Edges E_1 connect nodes corresponding to pairs of stackable real items, edges $E_2 = \{\{i, j\} \mid i \in V_1 \cup V_2, j \in V_2\}$ contain all pairs involving at least one dummy item. For an incoming item $i \in I^{in}$ and a fixed item $j \in I^{fix}$ an edge $\{i, j\} \in E_1$ exists if i can be stacked on j, i.e., if $s_{ij} = 1$. The costs c_{ij} for these edges are set to

$$c_{ij} := \begin{cases} 0, & \text{if } d_i \leq d_j \\ 1, & \text{otherwise.} \end{cases}$$

For two incoming items $i, j \in I^{in}$ an edge $\{i, j\} \in E_1$ exists if i can be stacked on j or vice versa, i.e., if $s_{ij} + s_{ji} \geq 1$. The costs c_{ij} for these edges are set to

$$c_{ij} := \begin{cases} 0, & \text{if } (s_{ij} = 1, d_i \leq d_j) \text{ or } (s_{ji} = 1, d_j \leq d_i), \\ 1, & \text{otherwise.} \end{cases}$$

The costs c_{ij} for edges connecting two real items i, j of the set V_1 are 0 if these items can be stacked without inducing an unordered stacking. If i and j can only be stacked leading to an unordered stacking, the cost c_{ij} is equal to 1. Furthermore, we set $c_{ij} := 0$ for all edges $\{i, j\} \in E_2$, i.e., all dummy items can be stacked at no cost. Such an edge corresponds to the situation that a stack is completely empty (two dummy items) or a stack contains a real item stored at the ground level without a stacked item on top (one real and one dummy item).

Consider the minimum-weight perfect matching problem defined on the graph G. Clearly, if a perfect matching does not exist, then also no feasible stacking solution exists. Otherwise a minimum-weight

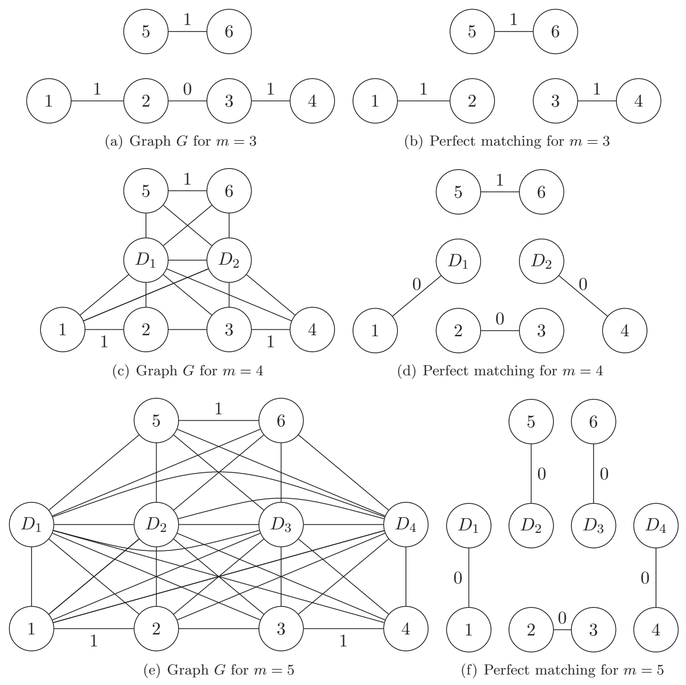

(a) Graph G for $m = 3$
(b) Perfect matching for $m = 3$
(c) Graph G for $m = 4$
(d) Perfect matching for $m = 4$
(e) Graph G for $m = 5$
(f) Perfect matching for $m = 5$

Fig. 2. Graphs and matchings for Example 2.

perfect matching arranges items in pairs ensuring the minimum possible number of unordered stackings. Indeed, due to the definition of c_{ij}, each unordered stack has cost 1, while stacks without unordered items or those containing only one item have cost 0.

Based on the solution to the minimum-weight perfect matching problem, we define a solution to the storage loading problem. For edges $\{i, j\} \in E_1$ in the matching we define the order of i, j in the stack as follows. If $i \in I^{in}, j \in I^{fix}$ or $i, j \in I^{in}$ are only stackable in one direction (i.e., $s_{ij} + s_{ji} = 1$), then they are stored in the unique possible way respecting the stacking constraints (i.e., if $s_{ij} = 1$, item i is stacked on item j). On the other hand, if items $i, j \in I^{in}$ are stackable in both directions (i.e., $s_{ij} = s_{ji} = 1$), then j is placed at the ground level and i is stacked on top if $d_i \leq d_j$; otherwise the reverse order is selected.

For edges $\{i, j\} \in E_2$ in the matching involving two dummy items, we introduce a completely empty stack; for edges $\{i, j\} \in E_2$ with one dummy and one real item, we place the real item on the ground level without any item on top.

Since the number of nodes is $2m$ and we assume that $m < n$, a minimum-weight perfect matching can be computed in $\mathcal{O}(n^3)$ time (cf. Gabow, 1973; Lawler, 1976). □

Example 2. Consider three instances with $n = 6$ items and $m \in \{3, 4, 5\}$, respectively. Let $s_{12} = s_{23} = s_{32} = s_{34} = s_{56} = 1$ and all other s_{ij} be zero. The delivery times are $d_1 = 4$, $d_2 = d_4 = 1$, $d_3 = d_6 = 2$ and $d_5 = 3$. In the basic graph for $m = 3$ shown in Fig. 2a items are connected if they can be stored together in a stack. Furthermore, the edges between two items have cost 0 if the items can be stacked without an unordered stacking and cost 1 if an unordered stacking occurs when stacking the items. In the case of Fig. 2a only item 2 can be stacked on item 3 without inducing an unordered stacking.

The basic graph shown in Fig. 2a is the graph that has to be considered if $m = 3$. Due to $2m = n$, no dummy nodes have to be introduced and a minimum-weight perfect matching with cost 3 is shown in Fig. 2b. This matching can be transformed into the stacking solution shown in Fig. 3a. For each stack the ordering of the two items is unique since they can only be stacked in one direction.

For $m = 4$ we introduce 2 dummy nodes D_1 and D_2 which are connected by edges of cost 0 to the 6 nodes corresponding to real items.

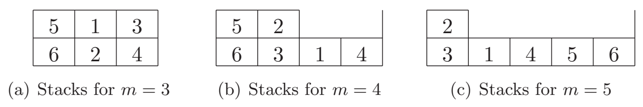


Fig. 3. Stacking solutions for Example 2.

Additionally, D_1 and D_2 are also connected, which enables empty stacks. The resulting graph is presented in Fig. 2c. Here, we do not label edges with 0 cost for clarity. In this situation, a perfect matching with cost 1 exists, see Fig. 2d. This matching can be interpreted as the stacking solution shown in Fig. 3b. Item 5 has to be stacked on top of item 6 since these items are in one stack and due to $s_{65} = 0$ item 6 cannot be stacked on top of item 5. For the second stack, items 2 and 3 are stackable in both directions, but putting 3 on top of 2 would induce an unordered stacking due to the delivery times ($d_2 < d_3$). Thus, item 2 should be stacked on top of 3.

For $m = 5$ we introduce the 4 dummy nodes D_1 to D_4 which are again connected by 0-cost edges to all nodes corresponding to real items; they are also interconnected by 0-cost edges. The graph is shown in Fig. 2e, a perfect matching with cost 0 is presented in Fig. 2f, and the corresponding stacking solution is shown in Fig. 3c. The only stacked items are 2 and 3, where again 2 is on top of 3 as this direction does not induce an unordered stacking.

Up to now, we assumed that one set I^{in} of incoming items is given and arbitrary stacking constraints s_{ij} have to be respected. If instead a sequence π^{in} or a sequence $(I^{in})_K$ of sets is given, the corresponding problems can be solved by the same algorithms as in Theorems 1 and 2. We simply use a modified stacking matrix $S' = (s'_{ij})$ with

$$s'_{ij} := \begin{cases} 1, & \text{if } s_{ij} = 1 \text{ and item } i \text{ does not arrive before item } j, \\ 0, & \text{otherwise,} \end{cases}$$

which guarantees that an item arriving at a later time is not placed underneath an item that arrives earlier. Thus, we have

Corollary 3. *Problems* $L, b = 2 \mid (I^{in})_K, s_{ij} \mid \#St$ *and* $L, b = 2 \mid (I^{in})_K, s_{ij} \mid \#SI^{>1}$ *can be solved in* $\mathcal{O}(n^{2.5})$ *time. Problem* $L, b = 2 \mid (I^{in})_K, s_{ij} \mid \#US$ *is solvable in* $\mathcal{O}(n^3)$ *time.*

4. Problems with stacking limit $b = 3$

In this section, we show that several problems with stacking limit $b = 3$ are strongly NP-complete.

Theorem 4. *The feasibility problem* $L, b = 3 \mid I^{in}, s_{ij} \mid -$ *is strongly NP-complete even for an empty storage area* ($I^{fix} = \emptyset$) *and transitive stacking constraints* s_{ij}.

Proof. We prove NP-completeness by a reduction from the strongly NP-complete problem EXACT COVER BY 3-SETS (X3C), see (Garey & Johnson, 1979). The idea of the proof is similar to that used in Garey and Johnson (1979) (Section 3.2.2) for proving NP-completeness of PARTITION INTO TRIANGLES.

An instance of X3C is given by a finite set X with $|X| = 3q$ for some integer q, as well as a collection C of three-element subsets of X. The decision problem asks whether C contains an exact cover of X, i.e., a subcollection $C' \subseteq C$ where each element of X is contained in exactly one element of C'.

For an instance of X3C we construct an instance of the stacking problem with $m = q + 3|C|$ stacks, stacking limit $b = 3$ and $n = 3(q + 3|C|)$ incoming items I^{in}. There are no items I^{fix} stored in the stacks. Note that $n = 3m$ and hence in each feasible solution all locations in the stacks have to be filled with items. We specify the instance of the stacking problem by a directed graph $G = (V, A)$, in which nodes V correspond to the items I^{in}, $|V| = n$, and arcs $(i, j) \in A$ exist if i can be stacked onto j (i.e., $s_{ij} = 1$).

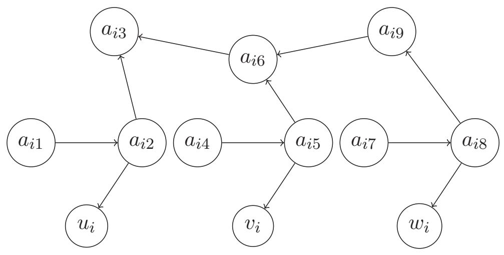


Fig. 4. Substitution graph.

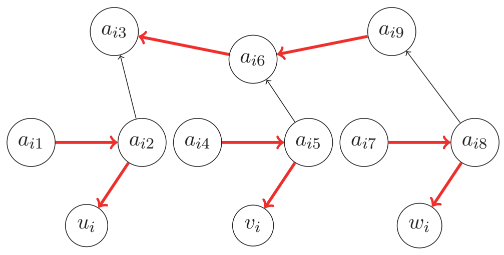


Fig. 5. Stacks if c_i is part of the cover. (For interpretation of the references to color in the text, the reader is referred to the web version of this article.)

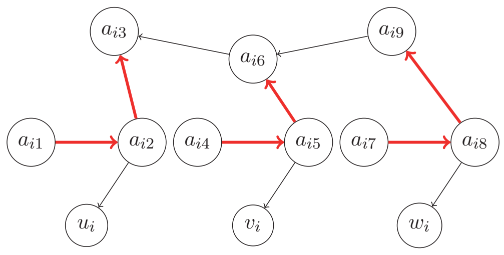


Fig. 6. Stacks if c_i is not part of the cover. (For interpretation of the references to color in the text, the reader is referred to the web version of this article.)

There are two types of nodes in V: main nodes (one node per element from X) and auxiliary nodes (9 nodes for each triple from C). Each triple $c_i = \{u_i, v_i, w_i\}$ defines a so-called *substitution graph* with 3 main nodes $\{u_i, v_i, w_i\}$, 9 auxiliary nodes $\{a_{i1}, \ldots, a_{i9}\}$ and the collection A_i containing the 11 arcs shown in Fig. 4. Note that the main nodes $\{u_i, v_i, w_i\}$ may belong to several substitution graphs, but there are no arcs between the a_{ij}-nodes belonging to different substitution graphs.

Thus, for the graph $G = (V, A)$ we have

$$V = X \cup \bigcup_{i=1}^{|C|} \{a_{ij} | 1 \le j \le 9\} \text{ and } A = \bigcup_{i=1}^{|C|} A_i.$$

In the following we show that C contains an exact cover of X if and only if the stacking problem has a feasible solution.

"$\Rightarrow$": Assume that X3C has an exact cover $C' \subseteq C$. If $c_i = \{u_i, v_i, w_i\} \in C'$, then in the stacking problem we build four stacks, each containing $b = 3$ elements: (a_{i1}, a_{i2}, u_i), (a_{i4}, a_{i5}, v_i), (a_{i7}, a_{i8}, w_i), and (a_{i9}, a_{i6}, a_{i3}). The corresponding paths within the substitution graph are drawn as thick red lines in Fig. 5. Note that the items corresponding to the main nodes of c_i are now stored and cannot be used in any other stack.

If on the other hand, c_i is not part of the cover C', the items in the corresponding substitution graph are stacked according to the thick red arcs shown in Fig. 6. Here, the three stacks (a_{i1}, a_{i2}, a_{i3}), (a_{i4}, a_{i5}, a_{i6}) and (a_{i7}, a_{i8}, a_{i9}) are built. In this case, the items corresponding to the main nodes of c_i are not included in any stack, while the items corresponding to all auxiliary nodes are stored.

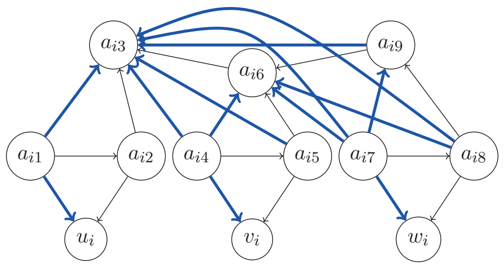


Fig. 7. Transitive substitution graph. (For interpretation of the references to color in the text, the reader is referred to the web version of this article.)

"$\Leftarrow$": Assume that conversely a feasible solution to the stacking problem exists. We construct a partition into triples which defines a solution to X3C.

First we note that due to the stacking constraints imposed by the arcs A, in any feasible solutions all main items are stacked at level 1, and the items at levels 2 and 3 correspond to auxiliary nodes.

Consider a stack with a main item at the ground level and an auxiliary item at level 2. Suppose that these items are u_i, a_{i2}; the cases of v_i, a_{i5} and w_i, a_{i8} are similar. Let G_i be the substitution graph that contains a_{i2}. We show that in this case there are four stacks

$$(a_{i1}, a_{i2}, u_i),\ (a_{i4}, a_{i5}, v_i),\ (a_{i7}, a_{i8}, w_i),\ (a_{i9}, a_{i6}, a_{i3}) \tag{1}$$

for the substitution graph G_i, and hence we include $c_i = \{u_i, v_i, w_i\}$ in the exact cover.

First observe that the stack with u_i and a_{i2} at levels 1 and 2 is of the form (a_{i1}, a_{i2}, u_i); thus there are 7 remaining auxiliary items of G_i, not counting a_{i1}, a_{i2}.

(i) If the item stacked on top of v_i does not belong to G_i and the same is true for the item stacked on top of w_i, then 7 auxiliary items of G_i cannot form full stacks containing $b = 3$ items.

(ii) If the item stacked on top of v_i does not belong to G_i , while the item stacked on top of w_i does, then (a_{i7}, a_{i8}, w_i) forms a stack in G_i, and the remaining 5 auxiliary nodes of G_i cannot form full stacks.

(iii) If the item stacked on top of v_i belongs to G_i, while the item stacked on top of w_i does not, then (a_{i4}, a_{i5}, v_i) forms a stack in G_i, and the remaining 5 auxiliary nodes of G_i cannot form full stacks.

Thus, in each of the above cases we get a contradiction to the assumption that there exists a feasible solution to the stacking problem, and the only feasible stacking is given by (1).

In the following we prove that already the special case of transitive stacking constraints s_{ij} is NP-complete by showing that the reduction from X3C also holds if the graph $G = (V, A)$ is transitive. For this purpose, we consider the transitive substitution graph shown in Fig. 7, where the transitive arcs are added as thick blue arcs.

We use similar arguments as before. Consider an item corresponding to the main node u_i included in a stack together with item a_{i2} (note that there does not exist a stack containing 3 items with item a_{i1} at level 2). Again it can be shown that there are four stacks (a_{i1}, a_{i2}, u_i), (a_{i4}, a_{i5}, v_i), (a_{i7}, a_{i8}, w_i) and (a_{i9}, a_{i6}, a_{i3}) for the substitution graph G_i, which means that $c_i = \{u_i, v_i, w_i\}$ should be included in the exact cover. Notice that property (i) remains the same; in (ii) we need the additional observation that a_{i7} cannot be stacked at level 2, immediately on top of w_i; similarly in (iii) we observe that a_{i4} cannot be stacked at level 2, immediately on top of v_i. $\square$

In the following we show that if we have no stacking constraints s_{ij}, but an additional weight or height limit per stack, the problem is also strongly NP-complete.

Theorem 5. *The feasibility problems* $L, b = 3 \mid I^{in}$*, weight-limit* $\mid -$ *with a weight limit per stack and* $L, b = 3 \mid I^{in}$*, height-limit* $\mid -$ *with a* *height limit per stack are strongly NP-complete even if the storage area is empty ($I^{fix} = \emptyset$).*

Proof. We prove NP-completeness of the first problem (the second problem can be tackled similarly) by a reduction from the strongly NP-complete problem 3-PARTITION (3-PART), see Garey and Johnson (1979). An instance of 3-PART is defined by a set $A = \{1, \dots, 3k\}$ of $3k$ elements and a bound $B \in \mathbb{N}$. Associated with the elements $i \in A$ are numbers a_i with $\sum_{i=1}^{3k} a_i = kB$ and $B/4 < a_i < B/2$. The decision problem asks whether A contains a partition $S_1, \dots, S_k$ such that $\sum_{i \in S_\lambda} a_i = B$ for $\lambda = 1, \dots, k$. Note, that by the definition of a_i each set S_λ must contain exactly three elements.

For an instance of 3-PART we construct an instance of the stacking problem with $|A| = 3k$ items and weights a_i for $i = 1, \dots, 3k$. There are no items I^{fix} stored in the stacks. We introduce $m = k$ stacks with stacking limit $b = 3$ and weight limit B per stack. We show that 3-PART has a feasible solution if and only if a feasible solution for the stacking problem exists.

"$\Rightarrow$": Assume that 3-PART has a feasible solution. Then the $k = m$ subsets can be interpreted as m stacks, each containing three items. These stacks are feasible since each stack respects the weight limit B and the stacking limit $b = 3$.

"$\Leftarrow$": Assume that conversely a feasible solution to the stacking problem exists. The m stacks have to be filled by three items per stack because the number of items is $3m$. Since the maximum weight of a stack is limited to B and the sum of the item weights is $B \cdot m$, each stack has a weight of B. Thus, the items of a stack can be interpreted as subsets with cardinality three for problem 3-PART, where for each subset the sum of the weights is B.

Note that the same reduction also applies to problem $L, b = 3 \mid I^{in}, \textit{height-limit} \mid -$ where the heights of the items take the role of the weights. Thus, this problem is strongly NP-complete as well. □

5. Problems with arbitrary stacking limit *b*

In this section, we consider an arbitrary stacking limit $b \geq 2$ which is given as part of the input. We show that the problem without stacking constraints is polynomially solvable if the objective is to minimize the transportation costs $TC(x, y)$, the number of items stacked above the ground level $\#SI^{>1}$, or a weighted sum of them. Additionally, we prove that problems $L \mid I^{in}, s_{ij} \mid \#St$ and $L \mid I^{in}, s_{ij} \mid \#SI^{>1}$ are polynomially solvable in the case that the stacking constraints s_{ij} define a total order.

At first we consider problem $L \mid I^{in} \mid w_1 TC(x, y) + w_2 \#SI^{>1}$.

Theorem 6. *Problem $L \mid I^{in} \mid w_1 TC(x, y) + w_2 \#SI^{>1}$ can be solved in $\mathcal{O}((n + mb)^3)$ time for any stacking limit b.*

Proof. We show that the problem can be solved as a minimum-weight matching problem in the bipartite graph $G = (V_1 \cup V_2, E)$ where V_1 corresponds to the set of incoming items I^{in} and V_2 contains all free positions in the storage area (here, a "position" means a pair consisting of a stack and a level). If the storage area is empty, we consider mb positions. If, on the other hand, the storage area is partially filled, we do not consider those positions that are already occupied with items from the set I^{fix}.

We introduce edges $\{i, p\}$ for all $i \in V_1$ and $p \in V_2$. The weight c_{ip} is defined as the cost of transporting item i from its origin O_i to position p multiplied by w_1. Additionally, if the position p is not at the ground level, we add the unit cost for a stacked item multiplied by w_2.

The bipartite graph G has at most $n + mb$ nodes, and the minimum-weight bipartite matching problem for it can be solved in $\mathcal{O}((n + mb)^3)$ time (cf. Gabow, 1976). Since in practical settings the stacking limit b is usually smaller than the number of items n, this complexity is polynomial.

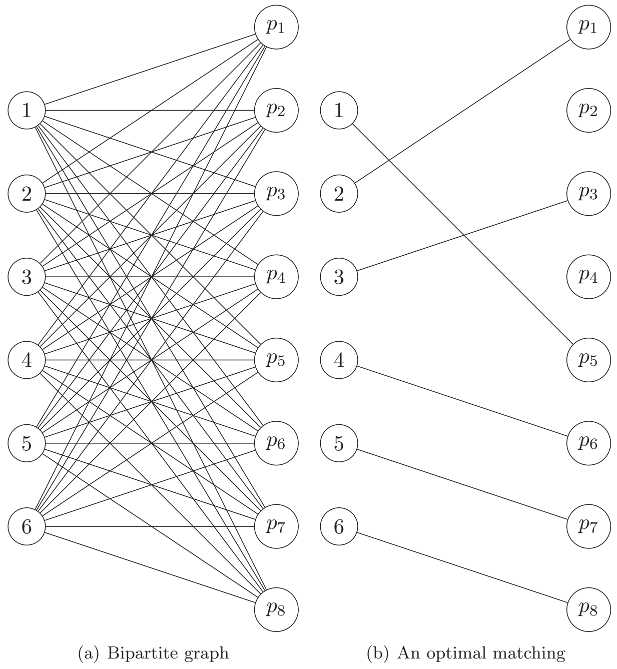


Fig. 8. Bipartite graph and an optimal matching for Example 3.

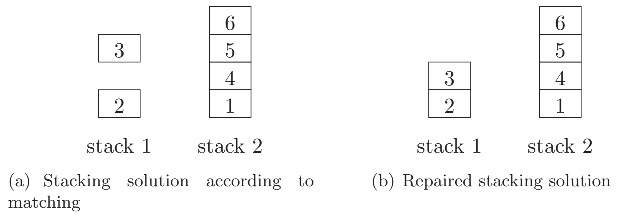

3
2
stack 1
6
5
4
1
stack 2
(a) Stacking solution according to matching
3
2
stack 1
6
5
4
1
stack 2
(b) Repaired stacking solution

Fig. 9. Original and repaired stacking solution for Example 3.

In an optimal solution to the matching problem it can occur that an item is stored in the kth level without an item stored in the $(k-1)$th level in the same stack. If $w_2 > 0$, then in an optimal solution all such k satisfy $2 \leq k \leq b$; if $w_2 = 0$, then $1 \leq k \leq b$. In either case, in the corresponding actual solution to the stacking problem, we can just lower all items that are stacked in the "air" until they are stacked on other items without increasing the cost. □

Example 3. Consider an instance with $m = 2$ stacks, stacking limit $b = 4$ and $n = 6$ items. The vertex sets have cardinalities $|V_1| = 6$ for items and $|V_2| = 8$ for free positions.

The corresponding bipartite graph is shown in Fig. 8a. Item vertices are numbered from 1 to 6, while position vertices are numbered from p_1 to p_8. Note, that the positions p_1 to p_4 belong to the first stack, while positions p_5 to p_8 belong to the second stack. Positions p_1 and p_5 represent ground level positions, the other positions represent non-ground levels for stacked items.

Assume that an optimal matching is the one shown in Fig. 8b. All items are assigned and positions p_2 and p_4 stay empty. This solution can be interpreted as the stacking solution shown in Fig. 9a. In that solution, in the first stack, level three is filled but level two is empty, which is not feasible. However, the solution can be repaired by lowering all items that are stored in the "air". In the solution shown in Fig. 9b the level of item 3 is changed from 3 to 2. The latter stacking solution has the same cost as the one shown in Fig. 9a since the costs for assigning an item to the second or third level in a stack are the same.

Now we consider the situation of special stacking constraints s_{ij}, which define a total order on all items, i.e., s_{ij} is transitive and for all $i \neq j$ we have $s_{ij} = 1$ or $s_{ji} = 1$. For example, this condition is satisfied if the s_{ij} are based on weight or length restrictions of the items.

Theorem 7. *Problems* $L \mid I^{in}, s_{ij}$ total order $\mid \#St$ *and* $L \mid I^{in}, s_{ij}$ total order $\mid \#SI^{>1}$ *can be solved in* $\mathcal{O}(n \log n)$ *time.*

Proof. Since the s_{ij} define a total order, all items can be compared. We define a comparator $\triangleq$ for items i and j based on the stacking restrictions s_{ij} as follows:

$$i \triangleq j := \begin{cases} i \approx j, & \text{if } s_{ij} = 1 \text{ and } s_{ji} = 1, \\ i \prec j, & \text{if } s_{ij} = 1 \text{ and } s_{ji} = 0, \\ i \succ j, & \text{if } s_{ij} = 0 \text{ and } s_{ji} = 1. \end{cases}$$

At first we consider the situation that the storage area is empty, i.e., $I^{fix} = \emptyset$. By sorting the items non-increasingly according to the defined comparator $\triangleq$, the list of items can always be transformed into a feasible stacking solution by iteratively filling the stacks from the bottom to the top according to the sorted list. This means that the first (the "largest") item is placed at the ground level in the first stack and all items up to the bth item are stacked on top. The $(b+1)$st item is at the ground level of the second stack and so on. In the following, we show that the stacking constraints are respected by this approach. Whenever an item i is stacked on another item j, we have $i \preceq j$ which implies that $s_{ij} = 1$ (for $i \preceq j$ we have to distinguish the situations $i \prec j$ and $i \approx j$, but according to the definition of the comparator for both cases $s_{ij} = 1$ holds). Since $s_{ij} = 1$ for all items i that are stacked on another item j, the stacking constraints are not violated by processing the ordered list. Obviously, the number of used stacks is minimized by this approach.

If the storage area is not empty (i.e., $I^{fix} \neq \emptyset$), we consider all items from I^{in} in non-decreasing order according to $\triangleq$ and fill up all partially filled stacks starting with the stack that has the smallest item $j \in I^{fix}$ on top (according to the comparator $\triangleq$). We fill the empty positions in this stack from the top to the bottom with the smallest feasible items from the set I^{in} (i.e., all $i \in I^{in}$ with $i \preceq j$). We then proceed with the stack with the second smallest item on top and so on. This strategy guarantees that incomplete stacks are filled as much as possible, up to the maximum level b. However, in the resulting solution still partly filled stacks may exist if there are not enough items smaller than the topmost item in a partial stack. Having processed all stacks containing items of I^{fix}, the problem reduces to the problem with an empty storage area discussed in the beginning of the proof. As a result, we either get a feasible solution or demonstrate that none exists. Again, the number of used stacks is minimized by this approach.

If a feasible solution to problem $L \mid I^{in}, s_{ij}$ total order $\mid \#St$ is found, a solution to problem $L \mid I^{in}, s_{ij}$ total order $\mid \#SI^{>1}$ with the minimum number of items stacked above the ground level can be obtained as before by moving as many items of the set I^{in} as possible from levels above the ground level to the ground.

Sorting the items of I^{in} and sorting the topmost items of I^{fix} (in the case that this set is not empty) requires $\mathcal{O}(n \log n)$ time, while filling up the stacks can be implemented in $\mathcal{O}(n)$ time. Thus, the problem can be solved in $\mathcal{O}(n \log n)$ time. □

Note that the ordered list in the proof defines a Hamiltonian path through the directed graph of the items with arcs (i, j) for $s_{ij} = 1$. If the s_{ij}-values do not define a total order, but the graph induced by s_{ij} contains a Hamiltonian path, this path can be transformed into a feasible stacking solution, adopting the same strategy as in the proof of Theorem 7. However, the existence of a total order or a Hamiltonian path is only a sufficient but not a necessary condition for the existence of a feasible stacking solution. For a feasible solution the graph has to be partitioned into at most m chains of length at most b, where each item is contained in exactly one chain.

Table 1
Summary of complexity results.

Problem	Reference	Complexity
$L, b = 2 \mid (I^{in})_K, s_{ij} \mid \#St$ or $\#SI^{>1}$	Corollary 3	$\mathcal{O}(n^{2.5})$
$L, b = 2 \mid (I^{in})_K, s_{ij} \mid \#US$	Corollary 3	$\mathcal{O}(n^3)$
$L, b = 3 \mid I^{in}, I^{fix} = \emptyset, s_{ij}$ transitive $\mid -$	Theorem 4	str. NP-complete
$L, b = 3 \mid I^{in}, I^{fix} = \emptyset,$ *weight-limit* $\mid -$	Theorem 5	str. NP-complete
$L, b = 3 \mid I^{in}, I^{fix} = \emptyset,$ *height-limit* $\mid -$	Theorem 5	str. NP-complete
$L \mid I^{in} \mid w_1 TC(x, y) + w_2 \#SI^{>1}$	Theorem 6	$\mathcal{O}((n + mb)^3)$
$L \mid I^{in}, s_{ij}$ total order $\mid \#St$ or $\#SI^{>1}$	Theorem 7	$\mathcal{O}(n \log n)$
$L \mid (I^{in})_2, I^{fix} = \emptyset, s_{ij}$ total order $\mid \#St$ or $\#SI^{>1}$	Theorem 8	$\mathcal{O}(n \log n)$

Example 4. As a small real-world example, we consider three types of items: 40, 42 and 45-feet containers, where the number denotes the length of the container in feet. In this example, 40-feet containers can be stacked on top of all container types, 42-feet container can be stacked on top of 42 and 45-feet containers, but 45-feet containers can only be stacked on 45-feet containers. By sorting the containers according to non-increasing lengths, we get a sequence of containers starting with 45-feet containers, continuing with 42-feet ones and ending with 40-feet containers. The order of containers of the same type can be arbitrary. By splitting this sequence into stacks of height b, the stacking constraints are respected for all stacks since items are stacked within their groups and potentially a 42-feet container is stacked on top of a 45-feet container and a 40-feet container is stacked on a 42-feet container, which is feasible.

Theorem 7 can be generalized to handle the situation where two incoming sets I_1^{in} and I_2^{in} are given. Then it is assumed that no item $i \in I_1^{in}$ can be stacked on top of an item $j \in I_2^{in}$, even if $s_{ij} = 1$.

Theorem 8. *Problems* $L \mid (I^{in})_2, I^{fix} = \emptyset, s_{ij}$ total order $\mid \#St$ *and* $L \mid (I^{in})_2, I^{fix} = \emptyset, s_{ij}$ total order $\mid \#SI^{>1}$, *where the stacks are initially empty, can be solved in* $\mathcal{O}(n \log n)$ *time.*

Proof. We use again the comparator $\triangleq$ that has been defined in the proof of Theorem 7. Consider first the items of the set I_1^{in}. Using the approach described in the proof of Theorem 7, create $\lfloor \frac{|I_1^{in}|}{b} \rfloor$ completely filled stacks, containing the smallest items and create one partly filled stack (if any) with $|I_1^{in}| \bmod b$ largest items. The problem with the remaining items of the set I_2^{in} reduces then to the problem considered in Theorem 7 with $I^{fix} = I_1^{in}$.

For the minimization of $\#SI^{>1}$, items of both sets I_1^{in} and I_2^{in} can be moved to the ground level. The complexity of the algorithm is dominated by the sorting step, which is again $\mathcal{O}(n \log n)$. □

Note that this algorithm cannot be generalized to solve problem $L \mid (I^{in})_3, I^{fix} = \emptyset, s_{ij}$ total order $\mid \#St$ with three incoming sets $I_1^{in}, I_2^{in}, I_3^{in}$; even the feasibility version of this problem is open.

6. Concluding remarks

In this work, we derived first complexity results for storage loading problems motivated by practical settings in container terminals or warehouses. The main assumptions are that the storage area is organized in fixed stacks with a limited height b and not every item may be stacked on every other item. Our results are summarized in Table 1.

Since in rail-road container terminals on average slightly above 1 up to 1.5 containers are stacked (cf. Ballis & Golias, 2002), in practice the stacking limit $b = 2$ may be sufficient. For this limit we have proposed efficient algorithms for minimizing $\#St$, $\#SI^{>1}$ or

$\#US$, which are applicable even in the presence of arbitrary stacking constraints s_{ij}. Additionally, if no stacking constraints are given, minimizing transportation costs is easy even for an arbitrary stacking limit b. Another important efficiently solvable case is characterized by stacking constraints s_{ij} defining a total order. Such constraints occur in practice if length or weight restrictions have to be taken into account.

On the other hand, problems with stacking constraints s_{ij} become NP-complete if the stacking limit is $b \geq 3$ or in the presence of height/weight limits. Such problems are typical, for example, for maritime terminals or warehouses, where higher stacks are common.

The most interesting problems for which the complexity status remains open are $L, b = 2 \mid I^{in}, s_{ij} \mid TC(x, y)$ involving transportation costs and $L, b = 3 \mid (I^{in})_K, I^{fix} = \emptyset, s_{ij}$ total order $\mid -$ with $K \geq 3$ sets of incoming items. It would also be interesting to study special situations for the matrix S based on a partial order, i.e., S is transitive, but not all items are comparable. For example, if arrival times a_i and departure times d_i are given, we may define $s_{ij} = 1$ if $a_i \geq a_j$ and $d_i \leq d_j$ to model that no reshuffling is allowed.

To conclude, we observe that the storage loading model studied in this paper is quite universal and context free. It is relevant to a broad range of scenarios: rail or maritime terminals, warehouses, storages, etc. Although the algorithms we propose are designed for rather special situations, they still might be useful as building blocks for more general problems with additional features and constraints typical for real-world applications.

Acknowledgement

This research was supported by the EPSRC funded project EP/K041274/1.

References

Avriel, M., Penn, M., & Shpirer, N. (2000). Container ship stowage problem: Complexity and connection to the coloring of circle graphs. *Discrete Applied Mathematics, 103*(1), 271–279.

Ballis, A., & Golias, J. (2002). Comparative evaluation of existing and innovative rail–road freight transport terminals. *Transportation Research Part A: Policy and Practice, 36*(7), 593–611.

Borgmann, B., van Asperen, E., & Dekker, R. (2010). Online rules for container stacking. *OR Spectrum, 32*(3), 687–716.

Boysen, N., Fliedner, M., Jaehn, F., & Pesch, E. (2013). A survey on container processing in railway yards. *Transportation Science, 47*, 312–329.

Caserta, M., Schwarze, S., & Voß, S. (2012). A mathematical formulation and complexity considerations for the blocks relocation problem. *European Journal of Operational Research, 219*(1), 96–104.

Dekker, R., Voogd, P., & van Asperen, E. (2006). Advanced methods for container stacking. *OR Spectrum, 28*(4), 563–586.

Delgado, A., Jensen, R. M., Janstrup, K., Høyer Rose, T., & Høj Andersen, K. (2012). A constraint programming model for fast optimal stowage of container vessel bays. *European Journal of Operational Research, 220*(1), 251–261.

Even, S., & Kariv, O. (1975). An $O(n^{2.5})$ algorithm for maximum matching in general graphs. In *16th Annual Symposium on Foundations of Computer Science* (pp. 100–112). IEEE.

Gabow, H. (1973). Implementation of algorithms for maximum matching on nonbipartite graphs. *PhD thesis*. Stanford, California: Department of Computer Science, Stanford University.

Gabow, H. (1976). An efficient implementation of Edmonds' algorithm for maximum matching on graphs. *Journal of the ACM, 23*(2), 221–234.

Garey, M., & Johnson, D. (1979). *Computers and intractability – A guide to the theory of NP-completeness*. San Francisco: Freeman.

Jaehn, F. (2013). Positioning of load units in a transshipment yard storage area. *OR Spectrum, 35*(2), 399–416.

Kang, J., Ryu, K. R., & Kim, K. H. (2006). Deriving stacking strategies for export containers with uncertain weight information. *Journal of Intelligent Manufacturing, 17*(4), 399–410.

Kim, K. H., Park, Y. M., & Ryu, K. R. (2000). Deriving decision rules to locate export containers in container yards. *European Journal of Operational Research, 124*(1), 89–101.

Lawler, E. (1976). *Combinatorial optimization: Networks and matroids*. New York: Holt, Rinehart and Winston.

Lehnfeld, J., & Knust, S. (2014). Loading, unloading and premarshalling of stacks in storage areas: Survey and classification. *European Journal of Operational Research, 239*(2), 297–312.