# An improved approximation algorithm for storage loading problems with stacking constraints

Tobias Oelschlägel[1], Sigrid Knust[1]

Institute of Computer Science, Osnabrück University, Osnabrück, Germany
{toelschlaege,sknust}@uni-osnabrueck.de

**Abstract**

We consider the problem of loading items into a storage area consisting of stacks which can contain up to $b$ items. Additionally, we are given directed stackability constraints describing which items can be stacked directly on top of each other. For the case of transitive stacking constraints, this problem is equivalent to partitioning a comparability graph into cliques of size at most $b$. For minimizing the number of used stacks, Moonen and Spieksma already presented a $(2 - \frac{1}{b})$-approximation; here, we describe an improvement of this algorithm to achieve an approximation ratio of $2 - \frac{3}{2b}$ for any $b \geq 3$.

**Keywords** : *storage loading, approximation, path partition*

## 1 Introduction

We are given a set of $n$ items that is to be inserted into a storage area consisting of stacks which can contain up to $b$ items. Moreover, hard stacking constraints have to be respected, which are encoded by a matrix $S = (s_{ij}) \in \{0,1\}^{n \times n}$ where $s_{ij} = 1$ if and only if item $i$ can directly be stacked on top of item $j$. A feasible solution is an assignment of the items to positions inside the stacks such that no gaps occur, no stack contains more than $b$ items, and an item $i$ is directly placed above item $j$ in the same stack only if $s_{ij} = 1$. Bruns et al. [1] showed that deciding whether there exists a feasible solution that uses at most $m$ stacks is possible in polynomial time if $b = 2$ and that it is $\mathcal{NP}$-complete if $b = 3$ and $S$ is transitive. In this paper, it is assumed that there is no restriction on the number of available stacks but instead the objective of minimizing the number of used stacks is considered.

Based on the stacking constraint matrix $S$, define the directed stacking constraint graph $G_S = (V, A)$ by $V := \{1, \ldots, n\}$ and $A := \{(i,j) \colon s_{ij} = 1\}$. A feasible solution using $m$ stacks then corresponds to a partition of the vertices $V$ into $m$ vertex disjoint directed paths of bounded length, i.e., paths containing no more than $b$ vertices (or $b-1$ edges). We will refer to a directed path consisting of $k$ vertices as a *$k$-path* (consequently, a single vertex is a 1-path). A feasible stacking configuration can be obtained from a path partition by assigning the items of the $i$-th path to the $i$-th stack. The order of the items inside the stacks is the same as in each path, e.g., the first item of the path is the top item and the last item is the bottom item.

The focus of this paper lies on transitive stacking constraints, which correspond to partial orders defined on the set of items. For example, a height $h_i$ and length $l_i$ could be associated with every item $i$, and then only smaller items are allowed to be placed on top of larger items, i.e., $s_{ij} = 1$ iff $h_i \leq h_j$ and $l_i \leq l_j$. Note that partial orders do not allow equivalent items, i.e., items $i, j$ with $s_{ij} = s_{ji} = 1$, due to antisymmetry; however, in the transitive case one can always place the items of a maximal set of equivalent items into an arbitrary order because the items of such set cannot be differentiated by the remaining items. Therefore, by modifying the stacking matrix $S$ to comply with such order of the items, it can be assumed that the stacking constraint graph is acyclic. For transitive stacking constraints $S$ consider also the undirected comparability graph $G_S^C = (V, E)$ where $E := \{\{i,j\} \colon s_{ij} = 1 \text{ or } s_{ji} = 1\}$. Every

directed path in $G_S$ corresponds to a clique in $G_S^C$ due to the transitivity, and conversely, every clique in $G_S^C$ is a totally ordered set of vertices and therefore it can be transformed into a directed path in the directed graph $G_S$ by arranging the items according to the total order. Consequently, finding a partition of $G_S$ into directed paths of bounded length is equivalent to finding a partition of $G_S^C$ into cliques of bounded size if the stacking constraints are transitive. In the following, the problem will be described from the perspective of partitioning the graph $G_S$ into directed paths, but one can always exchange "path in $G_S$" by "clique in $G_S^C$". For example, we want to note that removing any vertex from a directed path of length $k$ in $G_S$ (a clique of size $k$ in $G_S^C$) results in a directed path of length $k-1$ (a clique of size $k-1$).

## 2 Approximation algorithms

An approximation algorithm for minimizing the number of used stacks was given by Moonen and Spieksma [2] and it provides an approximation ratio of $2 - \frac{1}{b}$ for $b \geq 3$. More specifically, they considered a more general version of our problem where vertices have weights, but ours can be obtained by assuming that all vertices have the same weight. The approximation algorithm is based on computing a minimum path cover of the directed stacking constraint graph $G_S = (V, A)$. As $S$ describes a partial order, it can be assumed that $G_S$ is acyclic and therefore a minimum path cover can be computed efficiently by reducing it to the maximum bipartite matching problem. Given such a minimum path cover, the lengths of the paths are not restricted and if there are paths containing more than $b$ vertices they need to be cut into several shorter paths. If some path contains $k > b$ vertices, then it can be cut into $\left\lceil \frac{k}{b} \right\rceil$ paths containing at most $b$ vertices. Thus, the outline of the approximation algorithm is as follows:

1. Compute a minimum path cover $P_1, \ldots, P_w$ of $G_S$.
2. Cut each path $P_i$ into $\left\lfloor \frac{|P_i|}{b} \right\rfloor$ $b$-paths and one $r_i$-path where $r_i := |P_i| \bmod b$.

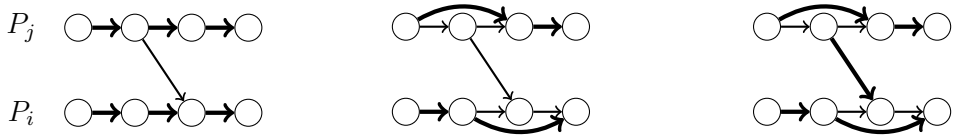
Now, let $\mathrm{opt}(S, b)$ be the minimum number of paths needed to cover $G_S$ with paths of length at most $b$. To bound the approximation ratio, first observe that $\mathrm{opt}(S, b) \geq \frac{n}{b}$ and $\mathrm{opt}(S, b) \geq w$. For each $i = 1, \ldots, w$ we introduce a binary variable $y_i$ that is 1 if cutting path $P_i$ results in a path of length smaller than $b$, i.e., $y_i$ is 1 if $r_i > 0$ and 0 otherwise. It then holds for the objective value of the approximate solution that

$$\begin{aligned}
\mathrm{apx} &= \sum_{i=1}^{w} \left\lceil \frac{|P_i|}{b} \right\rceil = \sum_{i=1}^{w} \frac{|P_i| - r_i}{b} + \sum_{i=1}^{w} y_i \\
&= \sum_{i=1}^{w} \frac{|P_i|}{b} + \sum_{i=1}^{w} (1 - \frac{r_i}{b}) y_i \\
&\leq \frac{n}{b} + w(1 - \frac{1}{b}) \leq (2 - \frac{1}{b})\mathrm{opt}(S, b)
\end{aligned}$$

It can be shown with an example that the approximation ratio is tight. In order to improve the algorithm, we will describe how to modify the minimum path cover and how to cut the long paths into smaller paths to ensure that there are at most $\frac{w}{2}$ paths consisting of a single vertex. This will then lead to an improved approximation ratio:

$$\mathrm{apx} \leq \frac{n}{b} + \frac{w}{2}(1 - \frac{1}{b}) + \frac{w}{2}(1 - \frac{2}{b}) = \frac{n}{b} + w(1 - \frac{3}{2b}) \leq (2 - \frac{3}{2b})\mathrm{opt}(S, b)$$

Let $R := \{P_i \colon |P_i| \bmod b = 1\}$ be the set of paths with remainder one, which will be called *short paths*. Instead of cutting the long paths of the cover into shorter paths, we try to match adjacent vertices of the short paths together. For example, if there exists an edge between two short paths $P_i, P_j \in R$, then by cutting $P_i$ and $P_j$ in the right way the adjacent vertices remain as 1-paths and can be merged into a single path (see Figures 1a to 1c). In Figure 1a




(a) Minimum path cover with two 4-paths (b) Cutting the 4-paths into 3-paths and 1-paths (c) Merging the two 1-paths to a 2-path

FIG. 1: Example for merging two 1-paths. Only the transitive reduction is shown. Simply cutting the long paths would result in four paths (for $b = 3$).

a minimum path cover with two 4-paths is shown. By cutting each of them simply into one 3-path and one 1-path, a solution consisting of 4 paths is obtained. To improve on this, the 4-paths can be cut in such a way that the vertices of the two 1-paths are adjacent (see Figure 1b). Afterwards, the 1-paths can be merged into a 2-path, which leaves a solution consisting of just 3 paths in total (see Figure 1c). There might not be any edges between vertices of short paths, so it is allowed to use edges that have only one endpoint in a short path. Also, to take care of a pathological case, it is allowed to cover three short paths with a single triangle (which results in a 3-path).

The goal is to find a vertex for every short path that can be moved into a 1-path that results from cutting it into paths of length at most $b$. Thus, we seek a set of vertex disjoint edges and triangles such that (i) every short path is covered by at least one edge/triangle, (ii) every edge contains vertices of different paths and at least one vertex of a short path, (iii) every triangle contains vertices of three different short paths. Given such a covering, we obtain a path partition by placing all vertices of the edges into a 2-path, placing all vertices of a triangle into a 3-path, and cutting the remaining paths into a minimum number of paths of length at most $b$. In the worst case, every short path is covered by an edge that ends in a path that has remainder two, i.e., for every short path that is eliminated, a new short path is created. This allows us to conclude that there are at most $\frac{w}{2}$ 1-paths in the final solution.

If such a covering does not exist, then there exists a special substructure, which can efficiently be partitioned in an optimal way. Every time such a substructure is found, the instance can be reduced and then the algorithm starts over with the remaining graph until a covering of the vertices is found. After computing a minimum path cover, the main procedure consists of finding the covering of the paths. Lemma 1 states that this is possible in polynomial time. As a minimum path cover can be computed in polynomial time and as there are $O(n)$ iterations in which a substructure can be removed from the graph, the total running time of the approximation algorithm is also polynomial in the size of the input graph.

**Lemma 1** *Let $G = (V, E)$ be an undirected graph and let $Q \cup V_1 \cup \ldots \cup V_k$ be a partition of $V$. In polynomial time one can find a certain substructure or determine a set of vertex disjoint edges and triangles such that*

1. *at least one vertex in $V_i$ is part of some edge or triangle for all $i \in \{1, \ldots, k\}$,*
2. *every edge covers vertices from two different parts of the partition,*
3. *every triangle covers vertices from three different parts of the partition.*

At last, we state the entire improved approximation algorithm:

1. Compute a minimum path cover $P_1, \ldots, P_w$ of $G_S$.
2. Determine $R := \{P_i \colon |P_i| \bmod b = 1\}$, and compute a covering of the paths in $R$ by vertex disjoint edges and triangles.
3. If such a covering does not exist, find the reducible substructure, partition its vertices optimally, and start over with the reduced graph.

4. Otherwise, place all vertices of the vertex disjoint edges into paths of length two, and all vertices of the triangles into paths of length three. Finally, cut each remaining path $P_i$ into $\left\lceil \frac{|P_i|}{b} \right\rceil$ paths.

### Acknowledgements

We are grateful for the constructive comments of two reviewers.

## References

[1] F. Bruns, S. Knust, and N. V. Shakhlevich. Complexity results for storage loading problems with stacking constraints. *European Journal of Operational Research*, 249(3):1074 – 1081, 2016.

[2] L. S. Moonen and F. C. R. Spieksma. Partitioning a weighted partial order. *Journal of Combinatorial Optimization*, 15(4):342–356, 2008.