Sorting with Complete Networks of Stacks

Felix G. König and Marco E. Lübbecke

Technische Universität Berlin, Institut für Mathematik, MA 5-1
Straße des 17. Juni 136, 10623 Berlin, Germany
{fkoenig, m.luebbecke}@math.tu-berlin.de

Preprint 036/2007

December 21, 2007
Revised: February 11, 2008

Abstract. Knuth introduced the problem of sorting with a sequence of stacks. Tarjan extended this idea to sorting with acyclic networks of stacks (and queues), where items to be sorted move from a source through the network to a sink while they may be stored temporarily at nodes (the stacks). Both characterized which permutations are *sortable*; but complexity of sorting was not an issue.
In contrast, given a complete, thus cyclic, network of $k \geq 2$ stacks, any permutation is obviously sortable. We ask how to do the sorting with a minimum number of *shuffles*, i.e., moves in between stacks. This is a natural algorithmic complement to the structural questions asked by Knuth, Tarjan, and others. It is the first time shuffles are considered in stack sorting—despite their practical importance.
We show that it is NP-hard to approximate the minimum number of shuffles within $\mathcal{O}(n^{1-\varepsilon})$ for networks of $k \geq 4$ stacks, even when the problem is restricted to complete networks, by relating stack sorting to MIN k-PARTITION on circle graphs (for which we prove a stronger inapproximability result). For complete networks, a simple merge sort algorithm achieves an approximation ratio of $\mathcal{O}(n \log n)$ for $k \geq 3$; however, closing the logarithmic gap to our lower bound appears to be an intriguing open question. Yet, on the positive side, we present a tight approximation algorithm which computes a solution with a linear approximation guarantee, using a resource augmentation to $\alpha k + 1$ stacks, given an α-approximation algorithm for coloring circle graphs.
When there are constraints as to which items may be placed on top of each other, deciding about sortability becomes non-trivial again. We show that this problem is PSPACE-complete, for every fixed $k \geq 3$.

1 Introduction

Stacks, as a fundamental data structure, play an important role in theoretical computer science, artificial intelligence, and combinatorial optimization. At the same time, stacks also model a wide range of applications in logistics and motion planning, where the access to items is restricted in a last-in first-out fashion.

We investigate a problem, in which k stacks are used for sorting. The k stacks form a directed network which we assume to be complete for most of the paper. A permutation of items to be sorted is given at a source node, and all items must arrive at a sink in correct order. Items may move along arcs in the network and may be stored temporarily on a stack at each node. When the network contains a cycle, any permutation can be sorted. Popping an item from one stack, moving it along an arc and then pushing it to the next stack is called a *shuffle*, and our goal is to minimize the number of shuffles needed to sort the given permutation.

Our Contribution. Our paper is the first primarily algorithmic view on stack sorting; it explicitly captures the essence of shuffles in sorting with a network of stacks. We prove that it is NP-hard to approximate the minimum number of shuffles within $\mathcal{O}(n^{1-\varepsilon})$ for $k \geq 4$ and any fixed $\varepsilon > 0$, even when the network of stacks is restricted to be complete, by relating STACK SORTING to the MIN k-PARTITION problem on circle graphs. For the latter problem we prove inapproximability within $\mathcal{O}(n^{2-\varepsilon})$ as an intermediate result which is of interest in its own. For the case of complete networks and $k \geq 3$, a simple merge sort algorithm computes an $\mathcal{O}(n \log n)$-approximation, but closing the gap to our lower bound appears to require significantly new insight into the problem (or into graph coloring, a we discuss). Still, we present an $\mathcal{O}(n)$-approximation algorithm which needs a resource augmentation to $\alpha k + 1$ stacks instead of only k, using an α-approximation algorithm for coloring circle graphs. We discuss that this is best possible in a certain sense. Furthermore, we prove that it is PSPACE-complete to decide whether a given permutation is sortable using a complete network of $k \geq 3$ stacks, when there are constraints as to which items may be placed on top of one another. We conclude with several challenging open problems.

Our results have direct consequences for various practical stacking problems from the operations research literature, e.g., [2, 3, 6, 9, 16], as well as for blocks world models in artificial intelligence [12, 17], among others.

Related Work

Stack Sorting. Knuth introduced the idea of stack sorting using the language of railway sidings [15]; he characterized permutations which can be sorted using k stacks in series. Tarjan extended these ideas to sorting with acyclic networks of stacks (and queues) [18]. Even and Itai considered the sortability of permutations using k parallel stacks [7]. They related this question to the problem of deciding k-colorability of a circle graph, which was proven to be NP-complete for $k \geq 4$ by Unger [19].

In all these papers (and those which followed), an item, once popped from a stack, may never be pushed back on it again. The point of interest has always been a characterization of which permutations can be sorted using a particular configuration of stacks. This (mathematically beautiful) point of view is surveyed by Bóna in [4]; he states, that “virtually nothing can be proved” for general networks of stacks.

Inspired by personal discussions about our work on this paper, Felsner and Pergel recently considered stack sorting from the perspective of extremal combinatorics [8]. Assuming that all items have to be moved to stacks before the first item may move to the sink, they consider instances in which an optimal solution has a maximum number of shuffles. They give good bounds on this number for different magnitudes of k.

Applications. In recent years, the operations research literature dealt with a number of practical applications involving stack sorting: Assigning incoming trains [6] or trams [3] to tracks of a switching yard or depot; parking buses in parking lots [9]; and stowage planning for container ships [2], to mention only a few. All of these ask whether it is possible to assign items to stacks such that items can be retrieved in a desired order without blocking each other. Even though shuffles are a natural part of real life stacking, researchers asked for sortability with parallel stacks, where shuffles are not an issue, instead of asking for sorting with few shuffles using a complete network of stacks, which would model many of the above applications more accurately.

König et al. have recently introduced a mathematical model and a heuristic for a particularly rich stacking problem which has the minimization of shuffles as objective. They model in great detail stacking problems occurring in the logistics of integrated steel production and in container terminal operation [16], which among other things include limited stack heights. PSPACE-completeness of deciding sortability is shown.

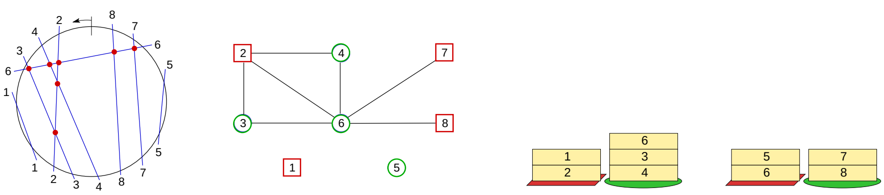
2 Problem Formulation and Relations to Graph Coloring

A formal definition of STACK SORTING is as follows. We are given a directed graph $G = (V \cup \{s,t\}, E)$ where s has no in- and t has no out-edges. To avoid trivialities, we require that any $v \in V$ is on an s-t-path in G, $|V| = k \geq 2$, and that G contains a cycle. A permutation π of items $1, \dots, n$ (the input sequence) is given at s, and has to be sorted, i.e., all items have to arrive at t in ascending order. Items may only move along arcs in G. When an item arrives at node $v \neq t$, it is stored on a stack S_v. A stack may be accessed on one end only, its *top*, so items may only leave in the reverse order they arrived. Naturally, items may only leave s in the order prescribed by π. Whenever an item moves along an arc (v,w) where $v \neq s$ and $w \neq t$, we say it is *shuffled*. The question is how to move items such that all items arrive at t in the correct order, using the smallest number of shuffles?

When $(s,v),(v,t) \in E$ for all $v \in V$, i.e., items can be moved from the source to any stack and from any stack to the sink, there is an interesting relationship between STACK SORTING and graph coloring. A graph coloring is an assignment of colors to the nodes of a graph. We call a coloring *proper*, if nodes which share an edge receive different colors; a k-coloring uses at most k colors. A *circle graph* is a graph the nodes of which can be drawn as chords of a circle such that two chords intersect iff the corresponding nodes share an edge. Even and Itai noted that deciding if a permutation π is sortable with k parallel stacks, i.e., $E = \{(s,v) : v \in V\} \cup \{(v,t) : v \in V\}$, is equivalent to deciding k-colorability of a circle graph the nodes of which are the items in π [7], which is hard for $k \geq 4$ [19]. Obviously, shuffles are impossible in the case of parallel stacks.

The class of circle graphs is equivalent to the so called *overlap graphs* [10]: Their nodes can be represented as intervals such that two nodes share an edge iff their corresponding intervals intersect but none of the two contains the other. With the latter representation, the correspondence of k-colorings to sorting with k parallel stacks becomes clear immediately: We define n intervals with unique start and end points in a discrete set of $2n$ points in time. The start points of the intervals are ordered corresponding to π. Immediately after the start of an interval, we insert the endpoints of all intervals corresponding to items i such that all intervals corresponding to items $1, \dots, i$ have already started, in ascending order. When two nodes do not share an edge, either their intervals do not intersect, which means the latter of the corresponding items arrives after the first has already left the buffer, or one of their intervals contains the other, which means that the corresponding items arrive at a stack in the reverse order they need to leave it. In both cases, the items can be put on the same stack. In any other case, the two items would block each other leaving the stack, so they may not be put on the same stack.

In a proper k-coloring of the nodes of the circle graph, the color of a node determines to which of the k stacks the corresponding item has to be moved to from s in order for all items to be able to arrive at t in correct order without shuffles. We now make the important observation that, if a proper k-coloring is impossible, *monochromatic* edges are unavoidable (edges with both endpoints in the same color), and this relates STACK SORTING to another coloring problem, the MIN k-PARTITION problem. The following example, cf. Fig. 1, illustrates the connection between circle graphs and STACK SORTING assuming G to be complete, i.e., $E = \{(s,v) : v \in V\} \cup \{(v,w) : v,w \in V\} \cup \{(v,t) : v \in V\}$. It also demonstrates that, quite counterintuitively, while a proper k-coloring of a circle graph does correspond to a solution to STACK SORTING without shuffles, asking for a coloring with fewest monochromatic edges is not the same as asking for few shuffles.



(a) (b) (c) (d)

Fig. 1. Optimally sorting $\pi = 24361875$ with one shuffle using $k = 2$ stacks; (a) shows the circle representation of the permutation, (b) the circle graph with a shuffle-optimal coloring of the nodes; (c) depicts the assignment of items to stacks before the shuffle, (d) the assignment after the shuffle and after moving the first four items out and all remaining items to the stacks.

The circle graph in Fig. 1(b) is clearly not 2-colorable. So obviously, at least one shuffle is necessary. The improper 2-coloring of the nodes represents the sorting of the permutation on two stacks shown in 1(c) and 1(d), which has exactly one shuffle and thus is optimal. The coloring has two monochromatic edges and is suboptimal in this sense: Changing the color of node 6 from green (circle) to red (square) would yield a coloring with just one monochromatic edge. However, the sorting of the permutation implied by this new coloring would have at least two shuffles.

At the core of the relationship between monochromatic edges in a coloring of a circle graph and the number of shuffles in sorting with complete networks of stacks lies the following observation: Suppose in a k-coloring of a circle graph, there is a color class with c nodes and many, say $c-1$, monochromatic edges. Consider a stack containing the corresponding c items. On one hand, all monochromatic edges could be incident to the item on top of the stack. In this case, shuffling this one item would, so to speak, resolve all $c-1$ monochromatic edges. On the other hand, each of the

$c-1$ monochromatic edges could connect two neighboring items on the stack—in this case, at least $\frac{c}{2}$ shuffles would be necessary. The point is, that in a coloring, the number of monochromatic edges for one color class does not depend on a certain ordering of its nodes, while on a stack, the order of its items is the single most important factor determining the number of shuffles necessary.

Yet, as we will see, the circle graph representation for sorting with complete networks of stacks is useful for proving hardness of approximation for STACK SORTING.

3 Hardness of Approximation

We show that it is NP-hard to approximate the minimum number of shuffles in STACK SORTING within a factor better than $\mathcal{O}(n)$. We start with the MIN k-PARTITION problem on graphs: one asks for deleting a minimum number of edges such that the remaining graph is k-colorable. We prove a strong inapproximability for this problem on circle graphs using ideas from [14] where the same result was shown for dense graphs.

Theorem 1. *Let G be a circle graph. For any $k \geq 4$, it is NP-hard to approximate the minimum number of monochromatic edges $\gamma(G,k)$ in a k-coloring of G within $\mathcal{O}(n^{2-\varepsilon})$.*

Proof. Let $I = (H,k)$ an instance of the k-COLORING problem for a circle graph $H = (V',E')$ with n nodes. Deciding if I is a yes-instance (which is equivalent to deciding if the minimum number γ^* of monochromatic edges in a k-coloring of H is 0) is NP-complete. We will construct an instance $J = (G,k)$ of MIN k-PARTITION where G is a circle graph with N nodes, such that approximating the minimum number $\gamma(G,k)$ of monochromatic edges in a k-coloring of G within $\mathcal{O}(N^{2-\varepsilon})$ is equivalent to deciding $\gamma^* = 0$, and thus to deciding I. In other words, our construction will create a quadratic gap in the possible optimal values of J, thus amplifying the hardness of deciding I to the hardness of approximating the optimal value of J.

Let $s := n^{\frac{2}{\varepsilon}-1}$. We construct $G = (V,E)$ as follows:

$$V := \{v_1, v_2, \ldots, v_s : v \in V'\},$$
$$E := \{(v_i, w_j) : (v,w) \in E';\ i,j = 1,\ldots,s;\ i \neq j\}.$$

In terms of the chord diagram of the circle graph, we obtain G from H by replacing each chord in H by s parallel chords in G. So G is a circle graph with $N := sn$ nodes and s^2m edges.

Now every k-coloring c of G can be transformed to a coloring c' in which all copies $v_i \in V$ of a node $v \in V'$ have the same color without increasing the number of monochromatic edges: For each $v \in V'$, pick the $v_i \in V$ with the fewest incident monochromatic edges and color all of v's copies in v_i's color. By this, it follows immediately that every optimal k-coloring c of G either has no or at least s^2 monochromatic edges: When there is at least one monochromatic edge, compute c' from c as described above. Let (v_i, w_j) be a monochromatic edge in c'. Since all s copies of w_j are neighbors of all s copies of v_i, there are at least s^2 monochromatic edges. Furthermore, we have

$$s^2 = n^{\frac{4}{\varepsilon}-2} = n^{\frac{2}{\varepsilon}(2-\varepsilon)} = N^{2-\varepsilon}.$$

Now suppose there was an algorithm computing an $\mathcal{O}(N^{2-\varepsilon})$-approximate solution to J with γ monochromatic edges. Then,

- $\gamma \in o(N^{2-\varepsilon}) \Rightarrow \gamma^* = 0 \Rightarrow H$ is k-colorable
- $\gamma \in \Omega(N^{2-\varepsilon}) \Rightarrow \gamma^* \geq 1 \Rightarrow H$ is not k-colorable.

Thus, such an algorithm would decide I. $\square$

We now establish some bounds relating monochromatic edges in the coloring of a circle graph and shuffles in solutions to the corresponding instance of STACK SORTING.

Lemma 1. *Let ℓ be a solution to an instance I of* STACK SORTING *with k stacks requiring L shuffles. Let $c_\ell : V \to \{1,\dots,k\}$ be the coloring of the corresponding circle graph $G = (V,E)$ obtained by assigning each node the color corresponding to the stack its item was first placed on, and let γ_ℓ denote the number of monochromatic edges in c_ℓ. Then,*

$$\gamma_\ell \leq (n-1) \cdot L.$$

Proof. From the correspondence between G and I it is clear, that for each monochromatic edge in c_ℓ, at least one of the items corresponding with its end points must be shuffled from its original stack in order to move both items to the output sequence. On the other hand, each item can only be incident to at most $n-1$ other items in G, so each shuffle can only "pay" for the need to shuffle items of at most $n-1$ monochromatic edges. $\square$

Lemma 2. *Let $c : V \to \{1,\dots,k\}$ be a coloring of a circle graph $G = (V,E)$ with γ monochromatic edges. Let I be the instance of* STACK SORTING *with a complete network of stacks corresponding to G. One can easily obtain a solution ℓ_c to I with $L_c = 2\cdot\gamma$ shuffles from c.*

Proof. The construction of ℓ_c happens in phases. A phase p ends whenever a prefix a_p of the identity permutation of all items not moved to the sink yet has been removed from the source, e.g., phase one ends immediately after item 1 has been removed from the source. During each phase, items are moved from the source to stacks as prescribed by c. When phase p ends, we move all items in a_p to the sink before continuing with the next phase: If the item needed next, say i, is not on top of its stack, we shuffle away all items above it to some arbitrary stack, move i to the sink, and then reverse all shuffles.

Note that for any item j which has to be shuffled in order to access i, we have $i < j$. On the other hand, when j was removed from the source, not all items m with $m < i$ had been removed from the source yet (otherwise i would have been moved to the sink in a previous phase). So from the correspondence between G and I, it is clear that $(i, j) \in E$, and this edge is monochromatic in c since i and j are on the same stack. So for each two shuffles in ℓ_c, there is a distinct monochromatic edge in c. $\square$

We are now ready to proof the main result of this section.

Theorem 2. *Approximating the minimum number L^* of shuffles in* STACK SORTING *within $\mathcal{O}(n^{1-\varepsilon})$ is NP-hard, even when restricted to complete graphs with a fixed number of $k \geq 4$ stacks.*

Proof. For an arbitrary circle graph G, let I denote the corresponding instance of STACK SORTING with a complete network of $k \geq 4$ stacks. Now suppose there was an $n^{1-\varepsilon}$-approximation algorithm for STACK SORTING. Then, for any instance, we can compute a solution ℓ with $L \leq n^{1-\varepsilon} L^*$ shuffles, where L^* denotes the number of shuffles in an optimal solution. As before, let c_ℓ be the coloring with γ_ℓ monochromatic edges obtained from assigning each node in G the color corresponding to the stack which the corresponding item was first placed on in ℓ. By Lem. 1, we have

$$\gamma_\ell \;\leq\; (n-1) \cdot L \;\leq\; n^{2-\varepsilon} L^* \;\leq\; n^{2-\varepsilon} L_{c^*} \;\leq\; 2 \cdot n^{2-\varepsilon} \cdot \gamma^*$$

where c^* denotes a coloring of G with the minimum number γ^* of monochromatic edges. The last inequalities follow from the fact that L^* is the minimum number of shuffles possible and Lem. 2.

Thus, an $\mathcal{O}(n^{1-\varepsilon})$-approximation algorithm for STACK SORTING immediately implies an $\mathcal{O}(n^{2-\varepsilon})$-approximation algorithms for MIN k-PARTITION on circle graphs.

□

Due to the generality of STACK SORTING, Thm. 2 has numerous consequences: The hardness of approximation immediately carries over to most applications involving sorting with stacks, and also to many blocks world planning models in artificial intelligence where table capacity, i.e., the number of stacks, is limited.

4 Approximation Algorithms for Complete Networks

We will now state our positive results. Even though their tightness w.r.t. our hardness result is unsatisfactory, we will discuss that they are the best we may currently hope for.

Fact 1 STACK SORTING *with* $k \geq 3$ *stacks can be done with* $2 \cdot (n \log n)$ *shuffles.*

This fact follows immediately from the application of a merge sort algorithm on stacks. An elaborate proof of this can be found in [8], where the authors also argue that there are instances for which $\Omega(n \log n)$ shuffles are needed when k is constant.

Remark 1. It follows from Fact 1, that in order to close the gap to the lower bound from Thm. 2, it would suffice to obtain an algorithm A which has a linear approximation guarantee only for instances, for which an optimal solution requires $o(\log n)$ shuffles. Returning the better solution of algorithm A and the mentioned merge sort would result in a linear approximation algorithm.

Closing the gap between hardness of sublinear approximation and the straight-forward $\mathcal{O}(n \log n)$ approximation appears to be very intriguing. One way to achieve this is at the expense of a resource augmentation.

Theorem 3. *There is an efficient algorithm which computes a solution with* $3n \cdot L^*$ *shuffles using* $\alpha k + 1$ *stacks where* L^* *denotes the minimum number of shuffles using k stacks, given an* α*-approximation algorithm for coloring circle graphs.*

Proof. Our algorithm proceeds in phases, the lengths of which are determined by repeatedly employing an α-approximation algorithm A for coloring circle graphs. Iteratively considering longer prefixes of π, we apply A to the circle graph defined by the current prefix, to determine the longest prefix which is still αk-colorable by A. We move the corresponding set P of items of the current phase to the first αk stacks according to the computed αk-coloring, meanwhile moving items to the output whenever possible. Since there may be items, which the coloring of the subgraph of this phase assumes to be movable to the output, but which we cannot actually move out yet, we use stack $S_{\alpha k+1}$ to store these items temporarily.

As a result, the items in P on each stack are now ordered from top to bottom on the first αk stacks and from bottom to top on $S_{\alpha k+1}$. We will now perform shuffles in order to have all these items on one stack, linearly ordered from bottom to top.

Let S_1 be the stack containing the set Q of all items from previous phases. We pick another arbitrary stack S_2 and merge all items in $X := P \cap (S_1 \cup S_2)$ onto a third stack S_3. We then merge all items in $Q \cup X \cup S_{\alpha k+1}$ onto S_2, now having all stacks ordered from top to bottom and S_1 empty. Finally, we merge all items from all stacks onto S_1 while moving items to the output whenever possible and, quite importantly, inserting the next item in π at the correct position, thus having sorted a part of π for which A could not find an αk-coloring. We have obtained a stack containing all items up to the current phase ordered from bottom to top.

Note that in one phase, we have shuffled each item at most three times. Hence, our algorithm needs $L \leq 3pn$ shuffles where p denotes the number of phases.

On the other hand, any solution to STACK SORTING requires at least one shuffle for each phase of our algorithm: Whenever the number of colors needed by A exceeds αk, the chromatic number of the circle graph exceeds k, thus at least one shuffle is necessary. Also, at the beginning of the next phase, our algorithm has moved the maximum number of items possible to the output and it may use all stacks available without restrictions. This is naturally the best possible situation for an optimal solution as well. Thus, the number of shuffles needed by an optimal solution is $L^* \geq p$, and $L \leq 3n \cdot L^*$.

The runtime of our algorithm is obviously dominated by the n calls to A. □

Remark 2. As the chromatic number of circle graphs cannot be determined exactly in polynomial time, a resource augmentation of at least one stack is unavoidable with our algorithm. This is rooted in the structure of circle graphs itself: In the stack assignment obtained from a proper coloring of the corresponding circle graph, the stacks do only remain sorted as needed by the algorithm as long as items are moved to the output immediately whenever possible. Thus, it is impossible to assign more than one color class to one stack: It may always happen, that items in an additional color class on the same stack keep items in the first color class from being moved to the output in time, thus destroying the linear order of items within one color class in the stack and possibly causing a non-constant number of additional shuffles in each phase of the algorithm. In that sense, our resource augmentation is best possible.

The relation to (improper) coloring circle graphs is our only algorithmic handle to approximating STACK SORTING (despite considerable efforts). Lower bounds for approximate colorings, in turn, classically rely on (a) maximum clique size ω, or (b) the number of vertices divided by the cardinality of a largest independent set. While for (b) a simple example shows that the bound is trivial (linear gap) for circle graphs, it is known for (a) that no better factor than $\log \omega$ can be obtained (this is mentioned in [1], citing a Russian paper by Kostochka; note that this falsifies Unger's claim of having obtained a 2-approximation [19]). In fact, the best known approximation factor for coloring circle graphs is $\alpha = \log n$ [5], and no improvement to a constant factor is possible without a new lower bound on the chromatic number of graphs.

Alternatively, abandoning colorings, one is tempted to characterize instances which need "few" shuffles (in the sense of Remark 1), yet, even deciding whether *no* shuffles are needed is NP-hard. On the other hand, if the permutation π avoids the pattern 1-2-3, no shuffles are needed if $k \geq 5$ (this is the result that every triangle free circle graph is 5-colorable, see again [1]). It becomes clear once more why circle graphs "frustrated mathematicians for some years" [11], and still continue to do so.

5 Stacking Constraints

We finally consider the generalization in which items may not be placed arbitrarily on top of others. An instance of STACK SORTING WITH CONFLICTS is an instance of STACK SORTING plus such constraints, which can be modeled as a directed graph D with node set $\{1,\dots,n\}$ such that an edge (i,j) in D signifies that item i may not be put directly on top of item j. This is a practically relevant extension [16], and sortability becomes a justified question again.

Deciding sortability in a more general setting is PSPACE-complete: In [16], initially, some items need to be placed on stacks in a well-defined configuration; in addition, there are height bounds on the stacks. Also, the number of stacks k is part of the input and this fact is crucial in the reduction proving hardness there.

We give a more elaborate construction, eliminating all these additional assumptions and prove the following significantly stronger result.

Theorem 4. *For any fixed $k \geq 3$, deciding whether there exists a feasible solution to an instance of* STACK SORTING WITH CONFLICTS *is PSPACE-complete.*

Proof. We give a reduction from a special case of CONFIGURATION TO EDGE shown to be PSPACE-complete in [13]: An instance $NCL = (G,C,e^*)$ of this problem is based on a 3-regular undirected graph $G = (V,E)$, node weights $c : V \to \mathbb{R}$ with $c \equiv 2$ and edge weights $w : E \to \mathbb{R}$ with $w(e) \in \{1,2\}$ $\forall e \in E$. C denotes a *feasible configuration*, given by an orientation of the edges in E such that the sum of the weights of edges pointing into a node is at least the node weight, i.e., $\sum_{e\in\delta^-(v)} w(e) \geq c(v)$ $\forall v \in V$, where $\delta^-(v)$ and $\delta^+(v)$ denote the sets of incoming and outgoing edges of a node $v \in V$ in C, respectively. $e^* \in E$ denotes a certain edge of the graph and the question is: Is there a sequence of feasible edge reversals such that the orientation of e^* is finally reversed?

We call edges with weight two *heavy*, all other edges *light*. G may only contain two types of nodes: Nodes with three heavy incident edges, called OR-nodes, and nodes with one heavy and two light incident edges, called AND-nodes.

The construction of an instance J of STACK SORTING WITH CONFLICTS for fixed $k \geq 3$ from an instance NCL of this special case works in three steps: First, we define basic gadgets consisting of items on two stacks for OR- and AND-nodes, respectively; then, we prove that the items of all gadgets can be put on any fixed number of $k \geq 3$ stacks without losing their properties crucial to the reduction; finally, we prove that we can number all items in the construction such that there exist a permutation π which forces the items into a configuration on stacks corresponding to the initial configuration C in *NCL*, and that a subsequent feasible reversal of e^* in *NCL* corresponds exactly to sortability of π. A more elaborate version of the proof for Thm. 4 is given in appendix A; at this point, we only state its main ideas.

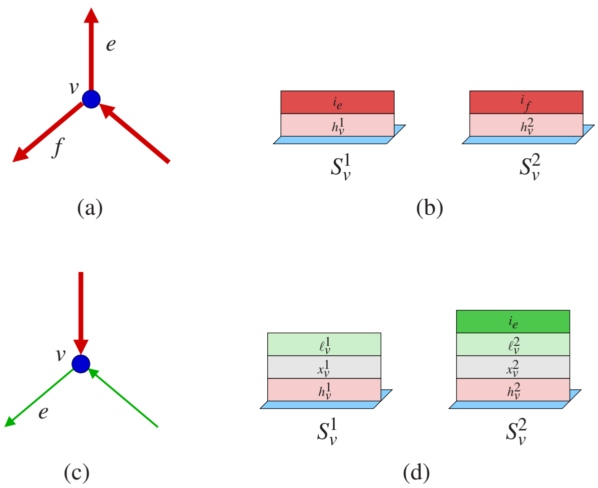
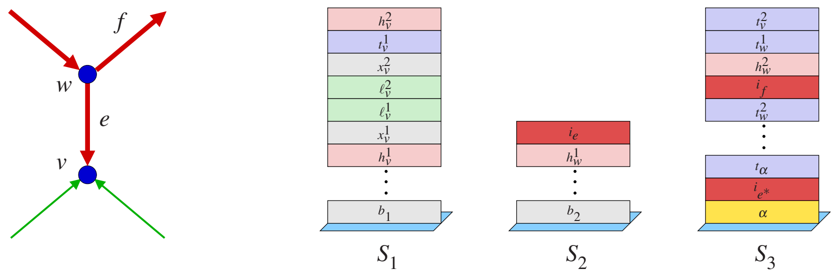


Fig. 2. Basic gadgets: Figs. (b) and (d) show stack representations of feasible configurations (a) and (c) at an OR-node and an AND-node respectively.

Fig. 2 shows the basic gadget for each OR- and AND-node. Each node $v \in V$ is associated with two stacks S_v^1 and S_v^2, and for each edge $e \in E$, we introduce an item i_e. A feasible configuration C in *NCL* corresponds to a configuration of items on stacks as follows: Item i_e is on stack S_v^1 or S_v^2 iff $e \in \delta^+(v)$ in C. We specify stacking constraints, such that only items corresponding to edges incident to a node v can be placed on the stacks corresponding to v. Also, we introduce some additional items, which may only be placed on the stacks of one single basic gadget (cf. Fig 2). Items i_e corresponding to heavy edges may only be placed on items h_v, items corresponding to light items only on items ℓ_v, and in any case it is required that $e \in \delta(v)$, i.e., e is incident to v. It is fairly easy to check, that feasible stackings on S_v^1 and S_v^2 correspond exactly with configurations in *NCL* which are feasible at v.

Fig. 3 shows how we can now assign all items from all basic gadgets to only three stacks while preserving their properties essential for the reduction. We separate the items of each stack of a basic gadget by items t with proper stacking constraints. Then we have one stack S_3 holding the items of stacks in the basic gadgets in reverse order.



(a) (b)

Fig. 3. Only using three stacks: Fig. (b) shows the stacking in which item i_e can be shuffled from S_w^1 to S_v^2. This corresponds to changing the orientation of e in Fig. (a).

In order to change the orientation of one edge, we shuffle items from S_3 to the other two stacks S_1, S_2, such that the stacks of two basic gadgets in between which we would like to shuffle an item i_e are exposed on top of S_1 and S_2 (cf. Fig. 3). Due to the stacking constraints specified, items from different basic gadgets never mix in the process.

The stack of the basic gadget containing i_{e^*} is at the bottom of S_3, in correct order. Directly underneath i_{e^*} is an item α, which clearly may only be moved, if the direction of e^* is feasibly changed before. With the help of some more additional items b, we can now define a numbering of the items and a permutation π in which α comes before all other items of basic gadgets, such that the constructed instance of STACK SORTING WITH CONFLICTS is sortable iff the orientation of e^* can be feasibly changed. □

6 Conclusions

Sorting with stacks is not a surprising connection between a fundamental data structure and a classic algorithmic theme. It is surprising however, that theory avoided the apparent need for shuffles—only sortability, not sorting itself has been considered so far. Our hardness results partially explain this lack of elegant and efficient algorithms.

Open Problems

Our work spawns some challenging open complexity issues. Hardness of approximation, i.e., Thm. 2, only holds for $k \geq 4$. Indeed, it is not even known whether polynomial time algorithms exist for $k = 2$ and $k = 3$.

There is still an annoying logarithmic gap between our inapproximability result and the best known approximation algorithm, which we only manage to close by resource augmentation. As pointed out in Sec. 4, the lower bound we use—the minimum number of monochromatic edges in a k-coloring of a circle graph—is not suited for a better result. Since (improper) coloring circle graphs is the only known handle to approximate STACK SORTING, what is an alternative lower bound on the number of shuffles?

Finally, also queues could be considered for intermediate storage of items (as e.g., Tarjan did). This is closely related to the so-called “midnight constraint” present in some applications, where all items have to be removed from the source, before the first item may be moved to the sink. The circle graphs then become permutation graphs which can be properly colored in polynomial time; thus, our hardness of approximation does not apply. But is this case really significantly easier?

References

1. A.A. Ageev. Every circle graph of girth at least 5 is 3-colourable. *Discrete Math.*, 195(1–3):229–233, 1999.
2. M. Avriel, M. Penn, and N. Shpirer. Container ship stowage problem: complexity and connection to the coloring of circle graphs. *Discrete Appl. Math.*, 103(1-3):271–279, 2000.
3. U. Blasum, M. Bussieck, W. Hochstättler, C. Moll, H. Scheel, and T. Winter. Scheduling trams in the morning. *Math. Methods Oper. Res.*, 49(1):137–148, 1999.
4. M. Bóna. A survey of stack-sorting disciplines. *Electron. J. Comb.*, 9(2), 2002.
5. J. Černý. Coloring circle graphs. *Electr. Notes Discrete Math.*, 29:457–461, 2007.
6. S. Cornelsen and G. Di Stefano. Track assignment. *J. Discrete Algorithms*, 5(2):250–261, 2007.
7. S. Even and A. Itai. Queues, stacks, and graphs. In *Proceedings of the International Symposium on the Theory of Machines and Computations*, pages 71–86. Academic Press, New York, 1971.
8. S. Felsner and M. Pergel. Sorting using networks of stacks and queues. Preprint, TU Berlin, Institut für Mathematik, 2008.
9. G. Gallo and F. Di Miele. Dispatching buses in parking depots. *Transp. Sci.*, 35(3):322–330, 2001.
10. F. Gavril. Algorithms for a maximum clique and a maximum independent set of a circle graph. *Networks*, 3(3):261–273, 1973.
11. M.C. Golumbic. *Algorithmic Graph Theory and Perfect Graphs*. Elsevier B.V., Amsterdam, second edition, 2004.
12. N. Gupta and D.S. Nau. On the complexity of blocks-world planning. *Artif. Intell.*, 56(2-3):223–254, 1992.
13. R.A. Hearn and E.D. Demaine. PSPACE-completeness of sliding-block puzzles and other problems through the nondeterministic constraint logic model of computation. *Theor. Comput. Sci.*, 343(1–2):72–96, 2005.
14. V. Kann, S. Khanna, J. Lagergren, and A. Panconesi. On the hardness of approximating max k-cut and its dual. *Chic. J. Theor. Comput. Sci.*, 1997, 1997.
15. D.E. Knuth. *The Art of Computer Programming, Volume 3: Sorting and Searching*. Addison Wesley Longman, second edition, 1998.
16. F.G. König, M.E. Lübbecke, R.H. Möhring, G. Schäfer, and I. Spenke. Practical solutions to PSPACE-complete stacking. In *Proceedings of the 15th ESA*, volume 4698 of *Lect. Notes Comput. Sci.*, pages 729–740. Springer, 2007.
17. N.J. Nilsson. *Principles of Artificial Intelligence*. Morgan Kaufmann, 1980.
18. R. Tarjan. Sorting using networks of queues and stacks. *J. ACM*, 19(2):341–346, 1972.
19. W. Unger. On the k-colouring of circle graphs. In *STACS 88, 5th Annual Symposium on Theoretical Aspects of Computer Science, Proceedings*, volume 294 of *Lect. Notes Comput. Sci.*, pages 61–72. Springer, 1988.

A Full Proof of Theorem 4

In this section, we give a more elaborate proof of Thm. 4 from Sec. 5.

Proof. As described in the short proof in Sec. 5, we give a reduction from an instance *NCL* of a special case of CONFIGURATION TO EDGE. The construction of an instance J of STACK SORTING WITH CONFLICTS for fixed $k \geq 3$ from *NCL* works in three steps:

Basic gadgets: We construct the initial gadgets for the nodes of G as follows: For each node $v \in V$, there are two stacks S_v^1, S_v^2. For each edge $e \in E$, there is an item i_e. We also add an additional item α, which will be underneath item i_{e^*}, the item corresponding to the edge the orientation of which we eventually want to reverse. When appropriate, we denote items corresponding to heavy or light edges by i_e^h and i_e^ℓ, respectively. For each $v \in V$, there are two additional items h_v^1, h_v^2 and for each $v \in V_{AND}$, there are four more additional items x_v^1, x_v^2 and ℓ_v^1, ℓ_v^2.

We denote the sets of heavy and light edges E_h and E_ℓ respectively and the sets of OR- and AND-nodes V_{OR} and V_{AND}. We specify the following stacking constraints, which on the one hand will turn out to be restrictive enough for the gagets to accurately mirror the behavior of OR- and AND-nodes, and on the other hand still permit items to be also stacked in reverse order as will be necessary later:

- h_v^1 and h_v^2 may only be put on i_{e_h} for $e_h \in \delta(v)$.
- For all $e \in E_h$, $e \neq e^*$, i_e may only be put on item h_v^1 and h_v^2 with $e \in \delta(v)$, $v \in V$.
- For each $v \in V_{AND}$, items $i_{e_1}^\ell, i_{e_2}^\ell \neq i_{e^*}$ corresponding to its incident light edges $e_1, e_2 \neq e^*$ may only be put on items ℓ_v^1, ℓ_v^2 with $e \in \delta(v)$, respectively.
- Items i_e may not be put on top of each other for all $e \in E$.
- Item i_{e^*} may only be put on top of item α.
- If e^* is a light edge, α may only be put on top of ℓ_v^1 or ℓ_v^2 with $e^* \in \delta^+(v)$ in C. Otherwise, α may only be put on top of h_v^1 and h_v^2 with $e^* \in \delta^+(v)$ in C.
- Items x_v^1, x_v^2 may only be put on top of h_v^1, h_v^2 and ℓ_v^1, ℓ_v^2 for all $v \in V_{AND}$.
- Items ℓ_v^1, ℓ_v^2 may only put on top of $i_{e_1}^\ell$, $i_{e_2}^\ell$, respectively, on top of x_v^1, x_v^2 and on top of each other for all $v \in V_{AND}$.

Furthermore, we assume for now that we can enforce for all stacks S_v^1, S_v^2, that if they are not empty, items h_v^1, h_v^2 are always bottommost in them, respectively. It will become clear how this is done in the next paragraph.

We now associate a configuration of items on stacks with a feasible configuration C in *NCL*: Item i_e is on stack S_v^1 or S_v^2 iff $e \in \delta^+(v)$ in C (cf. Fig. 2). Note that by the stacking constraints, only items corresponding to edges incident to a node v can be placed on the stacks corresponding to v. Also, each node's additional items can only be placed on its corresponding stacks.

If v is an OR-node, at most two items i_e can be placed on its stacks, so feasible stacks S_v^1, S_v^2 correspond with configurations which are feasible at v (cf. Fig. 2(a), 2(b)). If v is an AND-node, the item i_e corresponding to its heavy edge e can only be placed on h_v^1 and h_v^2. The only feasible way to put all this node's remaining items on the stacks is to put them all on one stack, say S_v^1. In any case, item x_v^1 or x_v^2 will be topmost in S_v^1, so no other item i_e can be placed on this node's stacks. Specifically, none of this node's two light edges may be directed outward. On the other hand, when one light edge is directed outward at v, its item must be on top of ℓ_v^1 or ℓ_v^2, say ℓ_v^1. Thus, ℓ_v^2 is topmost in the other stack, prohibiting v's heavy edge to be directed outward (cf. Figs. 2(c), 2(d)).

The possibility to put h_v^1, h_v^2 on i_{e_h} for $e_h \in \delta(v)$ and ℓ_v^1, ℓ_v^2 on top of $i_{e_1}^{\ell}$, $i_{e_2}^{\ell}$, respectively, does not harm our construction; its necessity will become clear in the next paragraph.

So a pair of feasible stacks S_v^1, S_v^2 corresponds with a feasible configuration C at $v \in V$ in NCL . Clearly, there is also a representation of each feasible configuration C in NCL as a feasible stacking in J. Consequently, moves of items i_e in between stacks of basic gadgets for different nodes correspond to feasibly reversing the orientation of a single edge in C.

Using only three stacks: The basic idea is to have one stack S_3, which initially contains all items in J, and two stacks S_1, S_2 in between which an item i_e can be moved whenever the reversal of the orientation of e in NCL is desired (cf. Fig. 3). For $k > 3$, we add $k-3$ items to the beginning of π which may not be stacked with any other items and which have to leave the stacks last, thus blocking all but three stacks.

In order to achieve this, we will allow whole configurations of items on stacks S_v^1, S_v^2 in the basic gadgets, call them partial stacks now, to be placed on top of each other: We introduce *separator items* t_v^1, t_v^2, which may be put on top of all items which can possibly be topmost in feasible configurations on stacks S_v^1, S_v^2, and vice versa (cf. Fig. 3). Also, all items t may be placed on top of one another.

We will, by the construction of π described further below, initially force all items, separated by separator items according to their partial stacks, on S_3 in reverse order. Say we need to change the orientation of an edge e currently oriented from w to v, and say i_e is currently among the items of partial stack S_w^1. We then move items from S_3 to the other two stacks until the items of partial stacks S_v^1, S_v^2 are topmost. We place items t_v^1 and t_v^2 on S_3 and conduct shuffles in between S_1 and S_2 until one of the two partial stacks topmost in S_1 and S_2, say S_v^2, is ready to receive i_e. We place t_w^1 and t_w^2 back on top of S_1 and S_2 and shuffle items as needed such that partial stacks S_w^1 and S_v^2 are topmost in S_1 and S_2 (cf. Fig. 3). We move t_w^1 and t_v^2 to S_3 and are now able to move i_e from S_w^1 to S_v^2, thus reversing the orientation of e in NCL.

Note, that by our stacking constraints, additional items belonging to different partial stacks, may never be put on top of one another, and thus will never mix in this process.

In order for the procedure described to be feasible, we still need to check that the stacking constraints we specified for the basic gadgets do not make it infeasible to stack their items on S_3 in reverse order. This is easily done by considering the seven possible partial stack configurations in reverse order: It is feasible to put

- x_v^1, x_v^2 on ℓ_v^1, ℓ_v^2, respectively
- ℓ_v^1, ℓ_v^2 on x_v^1, x_v^2, respectively
- ℓ_v^1, ℓ_v^2 on ℓ_v^2, ℓ_v^1, respectively
- ℓ_v^1, ℓ_v^2 on e_ℓ^1, e_ℓ^2, respectively for $e_\ell^1, e_\ell^2 \in \delta(v)$
- h_v^1, h_v^2 on e_h for $e_h \in \delta(v)$.

We also need to pay some special consideration to the partial stack containing α, call it S^α, since we need to guarantee, that α can only be moved once the orientation of i_{e^*} was feasibly changed. Thus, we place S^α bottommost, and contrary to all other partial stacks in correct order, in S_3 and forbid i_{e^*} to be placed on top of any separator element t: Now i_{e^*} can only be moved to a partial stack corresponding to e^*'s other node, which would correspond to feasibly changing the orientation of e^*.

Defining π and numbering items: In order to enable constraints as to which items may be put on stacks S_1 and S_2, two items, call them b_1, b_2, which may not be placed on top of each other or any other item, are first in π. From all partial stacks, only items h_1, h_2 may be put on top of them. Assume w.l.o.g. these items are placed on S_1, S_2. Next up are two items c_1, c_2, which may only be placed on top of b_1, b_2, and on top of which no other item may be placed, thus forcing all further items to be placed on S_3. Next, the partial stack containing α arrives bottom to top, finalized by its separator item t^α.

Next up in π are all items of all partial stacks S_v^1, S_v^2 in a configuration corresponding to the initial configuration C in *NCL*—the items of each partial stack arrive subsequently in reverse order, each partial stack preceded by its separator item t.

Item 1, which may be placed anywhere, but on top of which no item may be placed, is last in π, and items c_1, c_2 are numbered 2 and 3, so they can leave the buffer immediately. Naturally, item α is numbered 4: It can eventually leave the stacks iff *NCL* is a yes instance as argued. If we number the items below α in the same partial stack $5,\dots$ and all other items in the reverse order they appeared in π, they can all leave the buffer in ascending order.

Hence, for any fixed $k \geq 3$, we can efficiently construct an instance J of STACK SORTING WITH CONFLICTS with k stacks and at most $8|V|+c$ items from an instance $NCL = (G = (V,E), C, e^*)$ of PSPACE-complete CONFIGURATION TO EDGE, where c is a small constant and J has a solution iff *NCL* was a yes instance. $\square$