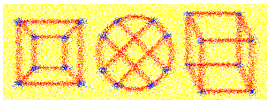

Journal of Graph Algorithms and Applications
http://jgaa.info/ vol. 30, no. 1, pp. 151–175 (2026)
DOI: 10.7155/jgaa.v30i1.3220

Minimizing the Height of Simple Tangles

Jakob Baumann *Ignaz Rutter*

Faculty of Computer Science and Mathematics, University of Passau, Germany

Submitted: March 2023 Accepted: May 2026 Published: June 2026

Article type: Regular paper Communicated by: M. Nöllenburg

Abstract. A *tangle* T of height h describes the ordering of n elements called *wires* on each of h levels. A pair of wires ab is called a *swap*. Between any two adjacent levels only the ordering of adjacent wires may swap and each wire may participate in only one swap. The set of all pairs that are swapped in this way defines a multiset $S(T)$ of swaps called the *swap list*. A tangle T *realizes* a given multiset S of swaps if $S(T) = S$. The problem Tangle Height Minimization asks for the minimum height of a tangle that realizes S.

In this paper, we study the tangle height minimization problem for *simple* tangles, whose swap list contains each swap at most once. First, we give algorithms with running time $O(n! \cdot 1.325^n)$ and $O((n+1)!)$ space and with running time $O(1.325^{hn})$ time but $O(n^2)$ space, thereby improving over a previous algorithm that uses $O(n! \cdot 1.618^n)$ time and space. Second, we show that the simple tangle height minimization problem can be recast as a graph orientation problem and propose an integer linear program based on this formulation. In an extensive experimental evaluation that considers several variants of each approach, we demonstrate that our exponential-time algorithms and the ILP are capable of solving instances with $n \leq 60$ in a reasonable period of time. Previous approaches can only handle instances up to $n \approx 20$.

1 Introduction

A *simple tangle* T of height h describes the ordering of n elements called *wires* on each of h levels. Between any two adjacent levels only the ordering of adjacent wires may swap and each wire may participate in only one swap. The set of all pairs that are swapped in this way defines a set $S(T)$ of swaps called the *swap set*. A tangle T *realizes* a given set S of swaps if $S(T) = S$. The problem Simple Tangle Height Minimization asks for the minimum height tangle that realizes S.

Tangles and the tangle height minimization problem were introduced by Olzsweski et al. [11] in 2018. They mainly focus on more general tangles, whose swap sets are multisets of swaps and the tangle realizing such a multiset has to swap every element as often as it appears. Their motivation

Funded by the Deutsche Forschungsgemeinschaft (German Research Foundation) under grant 412962361.

E-mail addresses: baumannjak@fim.uni-passau.de (Jakob Baumann) rutter@fim.uni-passau.de (Ignaz Rutter)

comes from the problem of visualizing so-called chaotic attractors [3,10]. Firman et al. [5,6] start a systematic study of the problem. Their focus is on the case where the tangles are not necessarily simple. They show that the tangle height minimization problem is NP-hard and give an improved exponential-time algorithm that runs in $O((2|L|/n^2+1)^{n^2/2} \cdot \phi^n \cdot n)$ time, where L is the list of swaps and $\phi \approx 1.618$ is the golden ratio. Their experimental evaluation indicates that the problem is difficult even for a low number of wires $n \leq 7$. They give a backtracking algorithm for simple tangles, that can optimize small instances ($n \approx 20$) in practice. Alistarov [1] studies a SAT-based approach to solving the tangle height minimization problem for non-simple tangles and tested them on the same instances as Firman et al [5,6]. Recently Firman et al. [4,5] strengthened the hardness result for non-simple tangles by showing that even the recognition problem is NP-hard, i.e., it is NP-hard to decide whether there exists a tangle that realizes a given swap list. Independently of Firman et al., Yamanaka et al. [14] show an equivalent result, namely that Ladder-Lottery Realization is NP-hard. By contrast, for *simple swap lists* S, where every swap occurs at most once, the information about tangles can be encoded in a graph G_S with vertex set $[n]$ and $i, j \in [n]$ are connected if and only if $ij \in S$. It is readily seen that S is realizable by a (simple) tangle with start ordering π if and only if G is a permutation graph that has a matching representation where one of the involved permutations is π. This can be tested in polynomial time [7], a result that was also rediscovered by Firman et al. [5,6].

For simple tangles, the complexity of the height minimization problem is still open, even though the problem itself dates back much further. Wang [13] studies the problem of sorting a permutation by swaps in the context of two-layer channel routing. Building on a technique by Sado and Igarashi [12], she shows that a parallel version of the bubble sort algorithm, which alternates between performing maximal swap sets at odd and even positions, always achieves a tangle height of at most n, and that this also yields an additive 1-approximation of the height. Firman et al. [5, 6] give an exponential-time algorithm for Simple Tangle Height Minimization that runs in $O(n! \cdot \phi^n)$ time (and space), where $\phi = \left(1+\sqrt{5}\right)/2 \approx 1.618$ is the golden ratio.

Contribution and Outline. We conduct a detailed study of Simple Tangle Height Minimization. In Section 3, we give two improved exponential-time algorithms. The first one has a running time of $O(n! \cdot \psi^n)$, and needs $O((n+1)!)$ space, where $\psi \approx 1.325$ is a constant. The second algorithm has a running time of $O(\psi^{hn})$, and needs $O(n^2)$ space, where h is the height of an optimal solution. In Section 4, we phrase the problem in terms of a graph orientation problem, which allows for a relatively simple way of formulating the problem as an ILP. Finally, in Section 5, we conduct a detailed experimental comparison that compares the efficiency of our exponential-time algorithm, our ILP formulation, and the exponential-time algorithm by Firman et al. The experiments show that our new algorithms significantly enhance the size of practically solvable instances from $n \approx 20$ to $n \approx 60$. From here onwards all swap lists are assumed to be sets (not multisets) and correspondingly all tangles are assumed to be simple.

2 Preliminaries

We denote the set $\{1, 2, \ldots, n\}$ by $[n]$. Following the intuition about tangles, we refer to elements $x \in [n]$ as *wires*. An *ordering* σ *of length* n is a total order relation $\prec_\sigma$ on $[n]$. Sometimes we write $\prec$ instead of $\prec_\sigma$, if it is unambiguous. We write an ordering σ as a sequence of numbers $ab\ldots z$ where $a \prec b \prec \cdots \prec z$. By $<$ we refer to the natural ordering.

Two wires $a, b \in [n]$ with $a \prec b$ are *adjacent* in σ if for any wire $c \in [n] \setminus \{a, b\}$ either $c \prec a$ or $b \prec c$. A *swap* is a set of two distinct wires, written as ab or $\{a, b\}$. A set S of swaps is *disjoint*

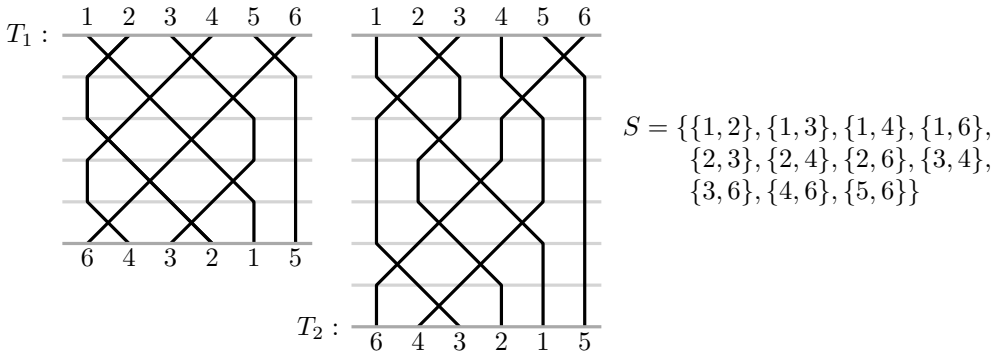


Figure 1: Two tangles T_1, T_2 on six wires that have the same (simple) swap set S.

if any two swaps $ab, cd \in S$ are disjoint, i.e. $\{a,b\} \cap \{c,d\} = \emptyset$. Two swaps that are not disjoint are *conflicting*. Given any ordering σ there is a unique swap set $S(\sigma) \coloneqq \{ab \mid a < b \wedge b \prec_\sigma a\}$ that induces the ordering $\prec_{S(\sigma)} \coloneqq \{(b,a) | (a < b \wedge ab \in S(\sigma)) \vee (b < a \wedge ab \notin S(\sigma))\} = \prec_\sigma$. Note that not every swap set induces a proper ordering on the natural ordering; if it does, we call the swap set *consistent*.

An ordering σ *supports* a disjoint swap set S if for every swap $ab \in S$ the wires a, b are adjacent in σ. For example $\sigma = 1234$ supports $S = \{12, 34\}$, but not $S = \{13\}$. Two orderings σ, τ are *adjacent* if $S(\sigma) \subseteq S(\tau)$ and σ supports $S(\tau) \setminus S(\sigma)$, or vice versa.

A *tangle* T of *height* h is a finite sequence $T = (\sigma_1, \ldots, \sigma_h)$ of orderings, such that σ_i, σ_{i+1} are adjacent for $i = 1, \ldots, h-1$ and $S(\sigma_i) \subseteq S(\sigma_{i+1})$. We call the first ordering σ_1 the *start ordering* and we assume, without loss of generality, that σ_1 is the natural ordering on n wires, since otherwise we can rename the wires. And we call σ_h the *final ordering*. The *swap sequence* $S_T = (s_1, \ldots, s_{h-1})$ of the tangle T is the sequence of the swap sets $s_i = S(\sigma_{i+1}) \setminus S(\sigma_i)$ between the layers $S(\sigma_i)$ and $S(\sigma_{i+1})$ for $1 \leq i \leq h-1$. Note that all s_i, s_j with $i \neq j$ are pairwise disjoint. See Figure 1 for an example.

The set $S(T)$ is the swap set of all swaps occurring in T. We say a tangle *realizes* a set S if it uses every swap in the set, i.e., $S = S(T)$. For a swap $ab \in S(T)$ the set $s(ab)$ is the unique s_i containing ab and $\mathrm{ind}(ab) = i$ is the unique *index* of ab. In a tangle $T = (\sigma_1, \ldots, \sigma_h)$ a swap ab *happens before* a swap cd if $\mathrm{ind}(ab) < \mathrm{ind}(cd)$.

A set S is *feasible* if there is a tangle T that realizes S. As Firman et al. [5, 6] show, a set is feasible if and only if it is consistent. We say a tangle T *realizes an ordering* σ if $S(T) = S(\sigma)$. The *minimum height* $h(S)$ of a swap set S is the minimum height over all tangles that realize S. In this work we want to find for a given set S a height-minimal tangle T realizing S, i.e. $h(S) = h(T)$.

3 A Faster Exponential Algorithm

In this section we improve the exponential-time algorithm of Firman et al. [5,6], which for a given swap set S computes a minimum height tangle in time $O(n!\phi^n)$, where $\phi \approx 1.618$ is the golden ratio. Note that they modeled tangles in terms of permutations instead of orderings, but this is obviously equivalent. Their algorithm recasts the problem as a shortest-path problem in a directed graph $G = (V, E)$ of exponential size. This graph G has one node for each of the $n!$ orderings of $[n]$. Let σ be such an ordering. If $S(\sigma) \not\subseteq S$, then σ cannot be reached from the start ordering with

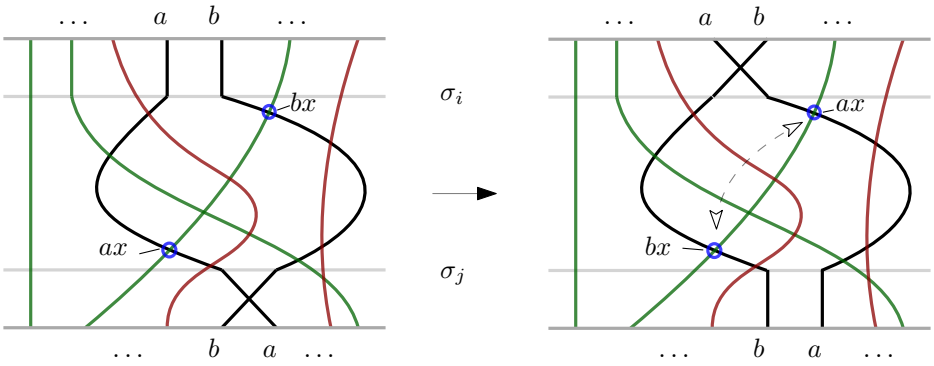


Figure 2: Performing a swap later in the tangle will never reduce the height of the tangle. We see that the green wires either neither cross a nor b or they have to cross both, hence performing the swap ab in an earlier layer induces a renaming but no structural changes. The red wires cannot exist, since they cross the same wire twice.

swaps from the set S. Otherwise, σ has an outgoing edge to every ordering τ that is adjacent to σ and where $S(\tau) \setminus S(\sigma) \subseteq S \setminus S(\sigma)$ holds. Clearly a shortest path from σ_1 to the final ordering $\sigma(S)$ corresponds to a minimum height tangle. The key observation of Firman et al. [5, 6] is that the out-degree of any vertex in this graph is $O(\phi^n)$ where $\phi \approx 1.618$ is the golden ratio. They thus compute a minimum-height tangle by first constructing the graph and then finding a shortest path using a BFS. Hence, their algorithm takes $O(n!\phi^n)$ time and space.

We improve on these bounds by constructing a subgraph of G for which the correspondence between shortest paths and minimum-height tangles still holds, but that has out-degree $O(\psi^n)$, where $\psi \approx 1.325$, yielding a running time of $O(n!\psi^n)$. We further show that we can compute optimal tangles with $O(n^2)$ space, slightly increasing the running time to $O(\psi^{hn})$, where h is the height of the tangle.

To this end, we prove that, for simple tangles, we can restrict our attention to tangles that apply in each step an inclusion-wise maximal swap set that is supported by the current ordering. More precisely, for an ordering σ and a swap set S, a set $S' \subseteq S \setminus S(\sigma)$ that is supported by σ is *maximal* with respect to σ if it is disjoint and for any swap $ab \in S \setminus (S' \cup S(\sigma))$, the set $S' \cup \{ab\}$ is not disjoint or is not supported by σ. We show that there exists an optimal tangle $T = (\sigma_1, \dots, \sigma_h)$ such that each s_i is maximal.

Theorem 1. *Let $T = (\sigma_1, \dots, \sigma_h)$ be a tangle with $S_T = (s_1, \dots, s_{h-1})$ its corresponding swap sequence. Let further $ab \in S(T)$ be a swap and $j = \mathrm{ind}(ab)$. Assume there is an index $i < j$ such that $s_i \cup \{ab\}$ is disjoint and supported by σ_i. For $i < k < j$ let s'_k be the the swap set obtained from s_k by exchanging in each swap the wires a and b. Then, the modified swap sequence $S'_T = (s_1, \dots, s_i \cup \{ab\}, s'_{i+1}, \dots, s'_{j-1}, s_j \setminus \{ab\}, \dots, s_h)$ induces a tangle T' for the same swap set.*

Proof: Note that for every swap $ax \in s_k$ with $i < k < j$ there must be a swap $bx \in s_l$ with $i < l < j$ in the tangle, since the wires a and b are adjacent in both σ_i and σ_j and the swap ax can only occur once. See Figure 2. Since T' does not change the structure of the tangle but simply switches names of a and b, the tangle T' still is valid. Note that σ_i and σ_j are unchanged, and therefore $S(T) = S(T')$. □

By Theorem 1, restricting G to the subgraph containing only the arcs that correspond to maximal swap sets, preserves the correctness of the algorithm of Firman et al. We now show that for any ordering σ of $[n]$ and any swap set S there are $O(\psi^n)$ swap sets that are maximal with respect to σ.

Lemma 1. *Let σ be an ordering of length $n \geq 2$ and S a swap set. Then, the number of maximal swap sets S' supported by σ with $S' \subseteq S \setminus S(\sigma)$ is in $O(\psi^n)$.*

Proof: We first prove by induction over the length of σ that the amount of maximal disjoint swap sets adjacent to an ordering of length n is $D(n) \leq D(n-2) + D(n-3)$ with $D(2) = 1, D(3) = D(4) = 2$. For this we simultaneously prove that $D(n) \geq D(n-1)$ holds.

First we see that the inductive hypotheses holds for $2 \leq n \leq 6$ by checking the cases. We give an outline for $n = 5$. From the first hypothesis we see that $D(n) \leq D(5) = D(5-2) + D(5-3) = 3$ should hold. We observe that there is the largest amount of different maximal swap sets if all swaps between the wires of σ are contained in S. Hence, for $n = 5$ we regard the swap set $\{12, 23, 34, 45\}$. There are exactly three maximal swap sets $\{12, 34\}, \{12, 45\}$ and $\{23, 45\}$, showing the first statement. Also see that $3 = D(5) \geq D(4) = 2$ for the second hypothesis. The other cases follow analogously.

Now we assume that for $2 \leq k \leq n$ and $n \geq 6$ the following two statements hold.

- $D(k) \leq D(k-2) + D(k-3)$ and
- $D(k) \geq D(k-1)$

Given an ordering σ' of length $n+1$ we know the statements hold for the orderings of length up to n arising from σ' by removing the last elements. Now we have to distinguish three cases how the wire at position $n+1$ behaves. We call the wire at position i just i for simplicity.

If $n(n+1) \notin S$ then obviously nothing changes from the ordering of length n to the one of length $n+1$, so $D(n+1) = D(n) \leq D(n-2)+D(n-3) \leq D(n-1)+D(n-2)$ and $D(n+1) \geq D(n)$ hold.

Secondly if $n(n+1) \in S, (n-1)n \notin S$ then we take any maximal adjacent swap set of the ordering from 1 to $n-1$ and add the swap $n(n+1)$ which is maximal again (and there are no other maximal swap sets not containing $n(n+1)$). So we have $D(n+1) = D(n-1) \leq D(n-3)+D(n-4) \leq D(n-1) + D(n-2)$ and $D(n) = D(n-1) = D(n+1)$.

In case $(n-1)n, n(n+1) \in S$ then there are two kinds of maximal swap sets. First we add $n(n+1)$ to every maximal swap set of the ordering from wire 1 up to $n-1$ which are $D(n-1)$ many, and secondly we add $(n-1)n$ and to every possible swap set from the ordering from 1 up to $n-2$, so $D(n-2)$ many. Hence, $D(n+1) = D(n-1) + D(n-2)$ and further $D(n) \leq D(n-2) + D(n-3) \leq D(n-1) + D(n-2) = D(n+1)$ hold.

Similar to Fibonacci's sequence we assume that we can find an x s.t. $D(n) = x^n$. We see

$$D(n) = D(n-2) + D(n-3) \Leftrightarrow x^n = x^{n-2} + x^{n-3} \Leftrightarrow x^3 = x + 1 \Leftrightarrow x^3 - x - 1 = 0.$$

We use Wolfram Alpha[1] to find the solution (only one real solution):

$$\psi := x_3 = \frac{\sqrt[3]{9 - \sqrt{69}} + \sqrt[3]{9 + \sqrt{69}}}{\sqrt[3]{18}} \approx 1.3247.$$

[1] (https://www.wolframalpha.com/)

So we see that for an ordering of length n there is a maximum of ψ^n adjacent inclusion-wise maximal orderings. □

We use this bound to speed up the search for a tangle of minimum height.

Theorem 2. *A minimal tangle can be computed in $O(n!\psi^n)$ time and $O((n+1)!)$ space or in $O(\psi^{hn})$ time and $O(hn)$ space.*

Proof: Constructing the graph where all edges correspond to maximal swap sets and applying a BFS immediately gives an algorithm with running time $O(n!\psi^n)$ and $O((n+1)!)$ space.

The space requirement can be alleviated by switching to a DFS as follows. Observe that, using Wang's algorithm [13], we obtain a tangle of height $h \leq n+1$ such that the optimal height is at least $h-1$. To test the existence of a tangle of height $h-1$ it thus suffices to run a single DFS with a depth limit of $h-1 \leq n$. To reduce the space requirement of the DFS we refrain from storing nodes that were already visited and stop at depth $h-1$. Further, we use an enumeration algorithm to generate all outgoing arcs of a node iteratively with amortized constant delay; we explain this below. In particular, we show that for every node σ, we can compute all outgoing arcs, of which there are $O(\psi^n)$ by Lemma 1, in $O(\psi^n)$ time and $O(n)$ space. Thus, for every node currently placed on the DFS-stack, we only store $O(n)$ information and compute the next outgoing arc in amortized constant time. As the stack holds at most $h-1$ elements at any time, our adapted DFS needs $O(hn)$ space, while increasing the running time to $O(\psi^{hn})$.

Now, given a node σ of n wires and a swap set S, we show how to generate all outgoing arcs, i.e., maximal swap sets $S_i \subseteq S$. Assume without loss of generality that $\sigma = \mathrm{id}_n$ and we number the n wires from left to right. We first partition the wires in *components* of consecutive elements with a swap in S, that is, maximal blocks of wires $C_i := \{l_i, l_i+1, \ldots, l_i+k_i\}$, such that for all $l_i \leq j < l_i + k_i$ we have $j(j+1) \in S$ is a swap. Observe that there is a unique partition of $[n]$ into components and that these are independent: the set of swaps performed in a component C_i does not influence the possible swaps in another component. All possible combinations can then be computed by generating all combinations of the components, which can be done in constant time per combination on average [8].

It thus remains to consider a single component. For simplicity, we assume that S consists of a single component, that is, for every $i \in [n-1]$ we have $i(i+1) \in S$. Let S_i be some maximal swap set of S. We call a wire that is not part of a swap in S_i *loose*. First, we consider configurations of maximal swap sets with exactly m loose wires. Note that there are $n-m$ wires that are swapped and between any two adjacent of the $(n-m)/2$ swapping pairs as well as in the first and the last position, there can be up to one loose wire, as two or more would contradict the maximality of the swap set. Thus, there are $\frac{n-m}{2}+1$ positions for the m loose wires, yielding exactly $\binom{(n-m)/2+1}{m}$ such configurations, which can be enumerated with average constant overhead per entry [9]. We do this for every $m \in [0, \lceil n/3 \rceil]$ where m has the same parity as n; see Figure 3 for an illustration. Thereby we generate every element in lexicographic order exactly once, with amortized constant delay. As there are $O(\psi^n)$ such elements by Lemma 1 and we visit every element only once, the overall running time for generating all outgoing arcs is bounded by $O(\psi^n)$. Together with the discussion above this shows the statement. □

Observe that for $h \in \Theta(n)$, we have $n!\psi^n \in o(\psi^{hn})$. The fact that the DFS has the potential to find a solution of size smaller than the result of Wang's algorithm without traversing almost the entire graph indicates a practical benefit of the DFS compared to the BFS, as we discuss more deeply in Section 5.

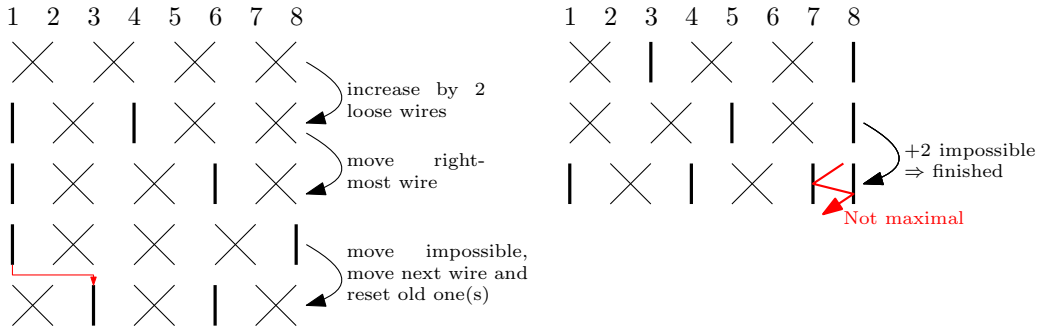


Figure 3: An example of how to algorithmically enumerate all possible maximal swap sets in Theorem 2 with $n = 8$. There are $D(8) = 7$ maximal swaps sets. The crossing lines symbolize swaps and the heavy vertical lines so-called loose wires.

4 Simple Tangle Height Minimization as a Graph Orientation Problem

The goal of this section is to recast the simple tangle height minimization problem as a graph orientation problem, where we seek an acyclic orientation of a graph that minimizes the height of the graph subject to certain conditions. We introduce the so-called swap graph, show how it relates to a tangle and give conditions under which we can obtain a tangle from an orientation of the swap graph. At the end of the section we derive an integer linear program (ILP) to find a minimal orientation of the swap graph and hence a minimal tangle. The running time of the ILP will be evaluated in Section 5.

4.1 The Swap Graph

Let S be a swap set. We define the *swap graph* $G = (V, E)$ over a swap set S as the graph with $V = S$ such that two different swaps ab and cd are adjacent if and only if $\{a, b\} \cap \{c, d\} \neq \emptyset$, i.e., if and only if the swaps share a wire.

An *orientation* O of G is a directed graph that is obtained from G by assigning a direction to each edge in G. An orientation is *acyclic* if it does not contain a cycle. The height $h(O)$ of an acyclic orientation is the length of a longest directed path in O.

Given a tangle T and its swap sequence $S_T = (s_1, \ldots, s_{h-1})$, we derive a corresponding orientation $O_T := \{(ab, ac) \mid \{ab, ac\} \in E, ab \in s_i, ac \in s_j, 1 \leq i < j \leq h - 1\}$ of G, which orients each edge of G from its swap that occurs first to its swap that occurs second. Observe that two swaps that share a wire belong to different sets in the swap sequence, and thus O_T is an orientation of G. It is readily seen that O_T is acyclic and, moreover $h(O_T) + 1 \leq h(T)$. The reason is that every step on a directed path moves to a swap that occurs later in the swap sequence. Since the sequence contains only $h(T) - 1$ elements, the bound follows.

We want to understand which orientations of G correspond to a tangle. For an edge $\{ab, ac\} \in E$, we define its *swap triple* as the swap set $\{ab, ac, bc\}$; note that possibly $bc \notin V$. We note that concepts similar to swap triples were first considered by Baumann [2] and subsequently by Firman et al. [4]. We call a swap triple *complete* if it is contained in V. We call an orientation O *a tangle orientation* if it is acyclic and satisfies the following two conditions, where $\prec$ denotes the start

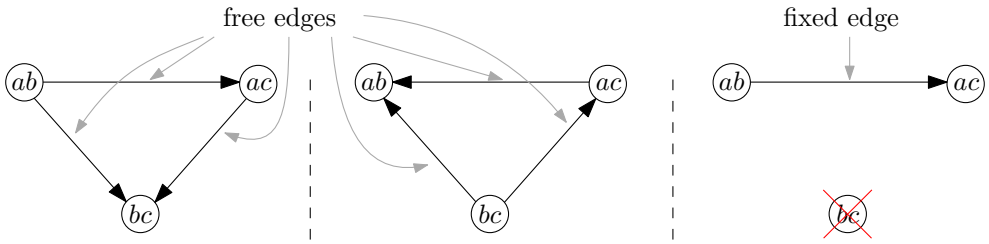


Figure 4: The order of the wires is $a \prec b \prec c$. We see the three possible arrangements of orientations of swap triples in a tangle orientation.

ordering.

(I) For each edge $\{ab, ac\} \in E$ with $a \prec b \prec c$ or $c \prec b \prec a$ where $bc \notin V$, O necessarily contains the edge (ab, ac).

(II) For each complete swap triple $\{ab, ac, bc\}$ where $a \prec b \prec c$, it must be true that either $(ab, ac), (ab, bc), (ac, bc) \in O$ or $(ac, ab), (bc, ab), (bc, ac) \in O$.

Lemma 2. *If T is a tangle, then O_T is a tangle orientation.*

Proof: We already observed that O_T is an acyclic orientation of $G = (V, E)$. The fact that O_T satisfies the first condition follows since if wire b starts out between wires a, c and b cannot swap with c, then b must swap with a before c can swap with a.

The second condition follows from the fact that the middle wire b first has to swap with a or with c. Say it swaps with a first. Then a has to swap with c, before b and c can become neighbors, allowing them to be swapped. This yields the first orientation; the second orientation is symmetric and is obtained if b swaps with c first. □

Thus, it is natural to restrict our attention to orientations that satisfy these two conditions. Observe that any two swap triples sharing an edge are already identical. Each incomplete swap triple has a unique orientation; we therefore call such edges *fixed*, whereas the edges of a complete swap triple are called *free*, and they may choose from two possible orientations, one of which is the reverse of the other. See Figure 4 for the different possible arrangements of a swap triple. Our next goal is to obtain a tangle from a tangle orientation.

Lemma 3. *For every tangle orientation O of a swap graph $G = (V, E)$ there exists a tangle $T = T(O)$ such that $O_T = O$.*

Proof: To construct the tangle, we iteratively remove the set of all sources of O. Let S_i denote the swap set that is removed in the ith iteration and observe that $k = h(O)$ is the last iteration.

We prove by induction on $h(O)$ that the swap sequence $S_O = (S_1, S_2, \ldots, S_k)$ defines a tangle of height $h(O) + 1$. To this end, we show that after removing the set of sources S_1 we again obtain an acyclic tangle orientation of the subgraph $G[V \setminus S_1]$ and its sub-orientation O' with start ordering $\sigma(S_1)$.

Consider the swap set S_1. Since all swaps of S_1 are sources, they are pairwise non-adjacent and therefore disjoint. We show that every swap of S_1 is supported by σ_1. Let $ac \in S_1$ and suppose for the sake of contradiction that there exists a wire b with $a \prec b \prec c$. As we only consider feasible sets, it follows that at least one of ab and bc is also a vertex of G. If just one of them exists, by

condition I it is connected to ac with a fixed edge pointing to ac, and ac could not be a source. Hence, $ab, bc \in V$ must hold. By condition II, there is an edge from exactly one of them to ac. In either case ac has an incoming edge, contradicting that it is a source. It follows that S_1 is a set of disjoint swaps that is supported by σ_1.

If $h(O) = 1$, this completes the proof. Otherwise, we show that $O' = O - S_1$ is an acyclic tangle orientation of G with respect to the new ordering $\sigma(S_1)$. Obviously we do not change any orientation, hence the remaining tangle orientation O' still is acyclic. Now consider any complete swap triple $\{ab, bc, ac\}$ with $a \prec b \prec c$. If none of the vertices was a source, properties I and II still hold. If one of them was a source, w.l.o.g. choose ab (ac can be excluded by the discussion above), then the new order becomes $b \prec_{\sigma(S_1)} a \prec_{\sigma(S_1)} c$ and (ac, bc) becomes a correctly oriented, fixed edge in O'. Let now (ab, ac) with $a \prec b \prec c$ be a fixed edge. Again only ab can be a source. If it is not a source, O' is still a tangle orientation since nothing changes. If it is, ab is removed. Since this removes the edge (ab, ac) there is nothing to be considered. We see that in every step, the height of the tangle orientation is decreased by one and therefore the induction is complete. □

The following theorem, which follows immediately from Lemmas 2 and 3, summarizes these findings. In particular, it follows that the problem of finding a minimum height tangle of a swap set S is equivalent to finding a tangle orientation of the swap graph G of S that has minimum height.

Theorem 3. *Let $G = (V, E)$ be the swap graph of a swap set S. Then the following hold:*

(i) For every tangle T that realizes S, O_T is a tangle orientation of G.

(ii) For every tangle orientation O of G there is a tangle T that realizes S with $O_T = O$.

4.2 Cycles in Swap Graph Orientations

Let G be a swap graph of a swap set S. To be a tangle orientation, an orientation O of G must be acyclic and satisfy conditions (I) and (II). The latter implies that each incomplete swap triple defines a fixed edge, whereas each complete swap triple offers a binary choice. To obtain a tangle orientation, we need to ensure that the binary choices offered by the complete swap triples do not create a cycle in O. We call an orientation of G that satisfies conditions (I) and (II) *a candidate orientation*. The following section shows that, rather than excluding arbitrary cycles, it suffices to exclude cycles of length 3.

Before we can prove this, we need a few lemmas. For their proof, we assume that the shortest cycle in O has length at least 4. We first observe that any minimal cycle C in O must be of the form $C = (ab, bc, cd, \ldots, ax)$ with each $a, \ldots, x \in [n]$ a different wire. So every wire participates in exactly two swaps that are adjacent in the cycle. We therefore use this notation for those cycles. The following lemma is very useful to understand when certain swaps must be contained in a feasible swap set.

Lemma 4. *Let S be a feasible swap set and $a \prec b \prec c$ for three wires $a, b, c \in [n]$. Then $ab, bc \in S$ implies $ac \in S$.*

Proof: Assume otherwise and let T be a tangle that realizes S. Assume that in T at first the wires a and b swap, the other case is symmetric. Then, after the swap, wire a is between the wires b and c. Hence, before b and c can be swapped, either swap ab or swap ac must occur. However, ab cannot occur a second time. Therefore, since S is feasible, the swap ac must exist. □

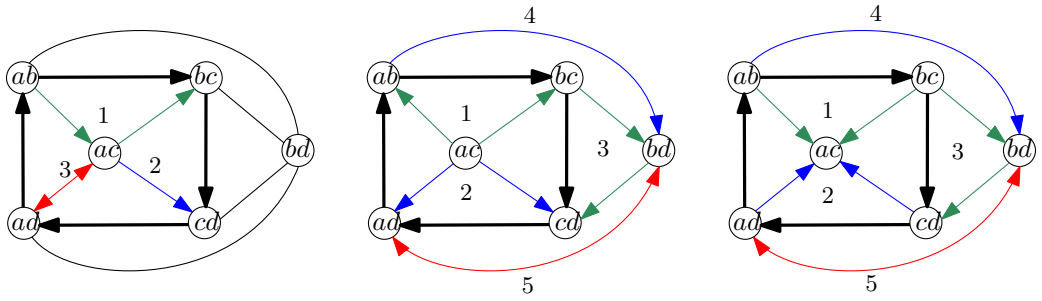


Figure 5: The three possible cases in the proof of Lemma 5, $m = 4$. The orders from left to right are $a \prec b \prec c$, $a \prec c \prec b$, and $b \prec a \prec c$. The reversed orders are analogous in each case. The edges are labeled according to the scheme in the proof of lemma and are not described again here.

The proofs of the next lemmas about cycles $C = (ab, bc, cd, \ldots, ax)$ are mainly extensive case distinctions on the order of the wires and the existence of certain swaps. To make everything clearer and shorter, we introduce the following conventions and notation.

We refer to Lemma 4 as condition (III) and we use the notation $ab \to bc$ to indicate that $(ab, bc) \in O$. We also use the following uniform coloring scheme for all images.

- Black and slightly thicker line segments are the given edges in the minimal cycle, the thin segments are edges that exist due to the definition of a swap graph, possibly directed by an assumption. For clarity we might omit some of them.
- Green for edges that are directed based on condition (II) for tangle orientations, i.e., they are contained in complete swap triples and one of the edges is directed.
- Blue if the reverse direction would create a shorter cycle.
- Purple for edges with fixed direction by condition (I) due to an inferred order and non-existence of a swap.
- Red is the edge that cannot be oriented in any direction without violating any condition or creating a shorter cycle.
- The numbers indicate the order in which the properties are inferred in the proof.

Lemma 5. *Let $G = (V, E)$ be a swap graph with candidate orientation O and let C be a minimum cycle of length $m \geq 4$. Then C does not contain two consecutive free edges.*

Proof: Let $C = (ab, bc, cd, \ldots, ax)$ and assume that $\{ab, bc\}$ and $\{bc, cd\}$ are both contained in a swap triple, i.e., $ac, bd \in V$.

We first consider the case $m = 4$. So let $C = (ab, bc, cd, ad)$ be this cycle. We distinguish the arrangement of wires in the abc-triple. See Figure 5.

1. Case $a \prec b \prec c$: See Figure 5 on the left. From condition (II) with order $a \prec b \prec c$ and $ab \to bc$ the directions $ab \to ac$ and $ac \to bc$ follow (green). We know that there cannot be a cycle of size 3, therefore the direction $ac \to cd$ follows, since otherwise (ac, bc, cd) is a cycle (blue). Now the edge (ac, ad) cannot be directed without generating one of the two shorter cycles (ab, ac, ad) or (ad, ac, cd) (red), which contradicts the assumption that C is a minimal cycle.

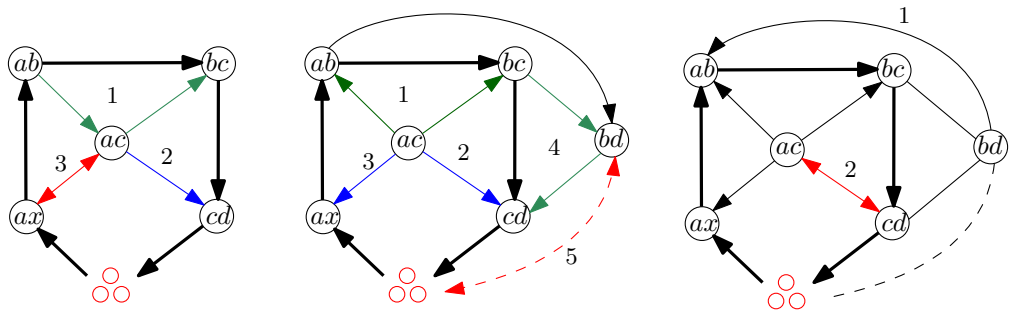


Figure 6: The first three cases of the proof of Lemma 5 for $m > 4$. The three red circles represent one or more additional swaps in the cycle. The four black thin arrows in the right image are to be reasoned exactly as in the middle one. The dashed line symbolizes the edge (bd, de) , where possibly $e = x$.

2. Case $a \prec c \prec b$: See Figure 5 in the middle. From the direction $ab \to bc$ with $a \prec c \prec b$ and condition (II) we infer $ac \to ab, ac \to bc$ (green, 1) and in order not to obtain one of the shorter cycles (ac, bc, cd) or (ac, cd, ad), we infer the directions $ac \to cd$ and $ac \to ad$ (blue, 2). By the orientation of the acd-triple we infer $d \prec a \prec c \prec b$, since either $c \prec a \prec d$ or $d \prec a \prec c$ holds, the former of which contradicts $a \prec c \prec b$. Thus, for the bcd-triple, with this order and condition (II), $bc \to bd, bd \to cd$ must hold (green, 3). To avoid the smaller cycle (ab, bc, bd) it must be $ab \to bd$ (blue, 4). Now the edge (ad, bd) cannot be directed without producing a shorter cycle (ab, bd, ad) or (cd, ad, bd), again contradicting the minimality of C (red).

3. Case $b \prec a \prec c$: See Figure 5 on the right. Again, due to the order $b \prec a \prec c$ the orientations $ab \to bc$, $ab \to ac, bc \to ac$ can be inferred by condition (II) (green, 1). Moreover, $ad \to ac$ and $cd \to ac$ must hold, because of the (ab, ac, ad) cycle and the (ac, cd, ad) cycle respectively (blue, 2). Due to the acd-triple, the order $b \prec a \prec c \prec d$ results, such that in the bcd-triple $bc \to bd$ and $bd \to cd$ apply (green, 3) and because of the shorter cycle (ab, bc, bd) then $ab \to bd$ must hold (blue, 4). Again, the edge $ad \to bd$ cannot be directed without generating a cycle (ad, bd, cd) or (ab, bd, ad) (red), which gives a contradiction to the minimality of C.

Due to symmetry, all possibilities have been considered and for the case $m = 4$ the statement is shown.

We now turn to the general case $m > 4$ and consider the same case distinction as before. This time the cycle is of the form $C = (ab, bc, cd, de, \ldots, ax)$. Depending on the size of the cycle, $e = x$ might hold. Let $ac, bd \in V$, i.e., the two swap triples abc and bcd are contained. In the figures, the new swap de and the remaining cycle are represented by the three red circles.

1. Case $a \prec b \prec c$: See Figure 6 on the left. Due to the order in the abc-triple and $ab \to bc$, it follows from condition (II) that $ab \to ac \to bc$ (green) holds. Now, because of the minimality of C, the edge must be directed as $ac \to cd$ (blue), otherwise (ac, bc, cd) is a shorter cycle. Either (ab, ac, ax) or $(ax, ac, cd, \ldots, ax)$ is a shorter cycle than C (red), which is a contradiction.

2. Case $a \prec c \prec b$: See Figure 6 in the middle. It follows from the order that all edges point

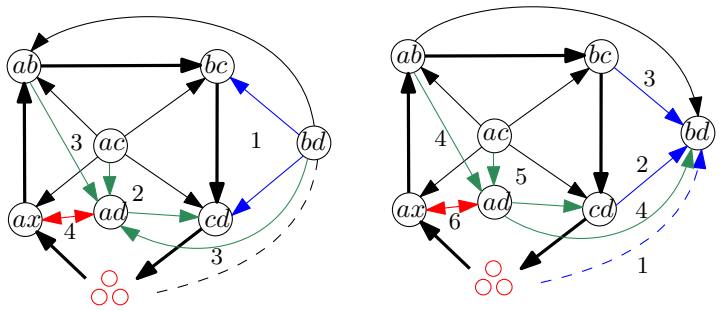


Figure 7: The last two subcases 2.c and 2.d of the second case for the proof of Lemma 5 for $m > 4$. The upper black directed arrow in both graphs is oriented this way by assumption in each case. The edges pointing away from ac derive exactly as in the picture of case 2.a. The justifications can be found in the color scheme at the beginning of the proof and in the respective proof section.

away from ac, so $ac \to ab, ac \to bc, ac \to cd, ac \to ax$ (the first two in green follow because of the abc-triple and since $ab \to bc$ holds; the latter two follow one after the other in blue to not create a shorter cycle). Now we further distinguish the following subcases:

(a) Case $ad \notin V \wedge ab \to bd$: See Figure 6 centered. Because of the fixed edge $ab \to bd$, now $d \prec a \prec c \prec b$ must hold and thus in the bcd-triple $bc \to bd, bd \to cd$ must hold (green, 4). Then the last dashed edge (red) cannot be directed correctly, since in each case a shorter cycle than C emerges.

(b) Case $ad \notin V \wedge bd \to ab$: See Figure 6 on the right. From the fixed edge $bd \to ab$ follows $a \prec d \prec b$. Consider the direction $ac \to cd$, which is now impossible, because to point that way $d \prec a \prec c$ would have to hold, which is incompatible with $a \prec d \prec b$. So the vertex ad would have to exist after all.

(c) Case $ad \in V \wedge bd \to ab$: See Figure 7, left. To avoid the shorter cycle (ab, bc, bd), $bd \to bc$ must be true and, because of the cycle (bd, bc, cd) the direction $bd \to cd$ follows (blue). Now we have a bcd-triple, so along with the initial condition $a \prec c \prec b$, the order $a \prec c \prec d \prec b$ must apply. Now we must direct the acd- and the abd-triple (green), which leads to a shorter cycle (ax, ab, ad) or $(ad, cd, \ldots, ax)$ at the red edge between ax and ad.

(d) Case $ad \in V \wedge ab \to bd$: See Figure 7 on the right. First, we can direct the blue arrows to not obtain a shorter cycle, starting with the dashed $de \to bd$ because of the cycle $(ab, bd, \ldots, ax, ab)$, followed by those in the bcd-triple also because of shorter cycles (bd, cd, de) for $cd \to bd$ and (bc, cd, bd) for $bc \to bd$. From the bcd triple, we then derive the order $a \prec c \prec b \prec d$ ($d \prec b \prec c$ can be excluded on the basis of the assumption $c \prec b$). Now, according to condition (II), in the abd-triple the directions $ab \to ad$ and $ad \to bd$ and in the acd-triple $ac \to ad$ and $ad \to cd$ follow (green). Thus in each case, the red edge (ax, ad) produces a shorter cycle $(ad, cd, \ldots, ax)$ or (ab, ad, ax).

3. Case $b \prec a \prec c$: See Figure 8 on the left (steps 1, 2 and 3). As for $m = 4$, all edges must point to ac, at first we derive $ab \to ac$ and $bc \to ac$ because of the abc-triple followed by the edge $ab \to bc$. Then, to avoid shorter cycles, we also conclude $ax \to ac$ and $cd \to ac$. We further distinguish the following cases:

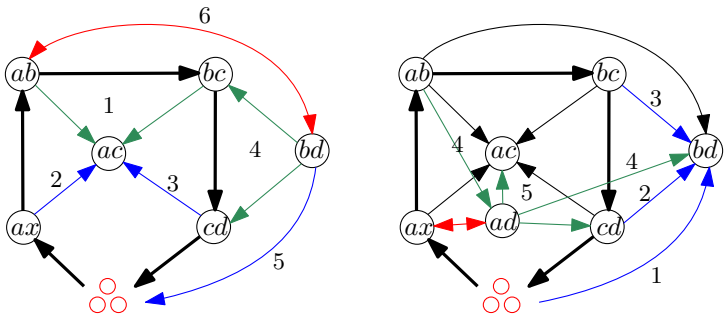


Figure 8: Cases 3.a and 3.b of the proof of Lemma 5. The black arrows directed to ac on the right follow exactly as on the left.

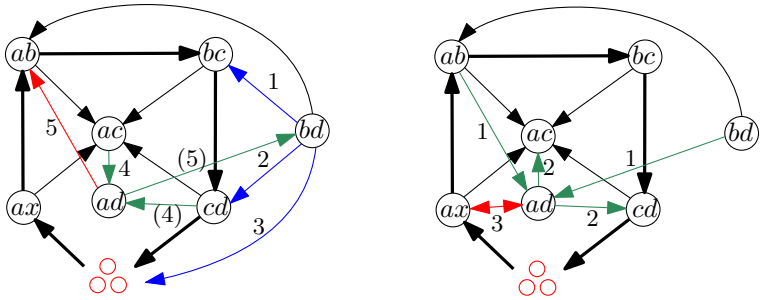


Figure 9: The last two subcases 3.c.i and 3.c.ii of the proof in Lemma 5. The black arrows pointing to ac result as in case 3.a above.

(a) Case $ad \notin V$: See Figure 8 on the left. Due to $cd \to ac$ being fixed $b \prec a \prec d \prec c$ holds ($c \prec d \prec a$ can be excluded after assuming $a \prec c$). By this order and $bc \to cd$, the bcd-triple can be directed (green) with $bd \to bc, bd \to cd$. To get no shorter cycle $(bd, cd, \ldots, bd)$ the swap bd must be pointing towards the remaining cycle (blue) $bd \to de$. Now the edge between ab and bd cannot be oriented correctly, because either the $(ab, bd, \ldots, ax)$-cycle arises, which is shorter, or ad would have to exist, in order to direct the edge $bd \to ab$ despite the order $b \prec a \prec d \prec c$ (red).

(b) Case $ad \in V \wedge ab \to bd$: See Figure 8 on the right. Clearly the edge $de \to bd$ must be oriented this way, otherwise there is already a smaller cycle $(ab, bd, \ldots, ax)$. To further avoid the cycles (cd, de, bd) and (bc, cd, bd) the other two blue edges in the bcd-triple are directed to bd resulting in $cd \to bd, bc \to bd$, from which we infer the order $d \prec b \prec a \prec c$ by the bcd-triple. Now we can orient the abd- and the acd-triples and obtain the directions $ab \to ad, ad \to bd$ and $ad \to ac, ad \to cd$ (green) and come to the conclusion that (ax, ad) cannot be oriented (red) without producing a shorter cycle $(ad, cd, \ldots, ax)$ or (ab, ad, ax).

(c) Case $ad \in V, bd \to ab$: See Figure 9 on the left. We see that the three blue directions $bd \to bc, bd \to cd$ and $bd \to de$, arise immediately from top to bottom to not create a shorter cycle. Now we can conclude from the bcd-triple that $b \prec d \prec c$ holds (where $c \prec d \prec b$ can be excluded because of the assumption $b \prec c$), which again yields

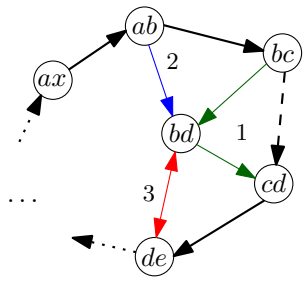


Figure 10: The first case of the proof of Lemma 7. The thick solid edges are fixed, the dashed ones are free, and the dotted ones can be either. We can assume that one edge is free because of the previous Lemma 7 . The green edge can be directed because of the swap triple, the blue to not produce a smaller cycle, but in the end the red one always produces a smaller cycle.

two final cases:

i. $a \prec d$: Because of $b \prec a \prec c$ the order is $b \prec a \prec d \prec c$. From this arrangement, the green directions $ad \to ab$ and $ac \to ad$ can be inferred for the abd- and acd-triples, resulting in a shorter cycle (ab, ac, ad).

ii. $d \prec a$: Here, the order is $b \prec d \prec a \prec c$. See Figure 9 on the right. Again, by the order, the abd- and the acd-triples can be arranged with $ab \to ad, bd \to ad$ and $ad \to ac, ad \to cd$ as indicated in green. In red, we see that the edge between ax and ad cannot be directed without producing the shorter cycle $(cd, \dots, ax, ad)$ or the cycle (ab, ad, ax).

All other cases are symmetric to the ones shown and result in a contradiction to the assumption. Hence the statement follows. □

Now we further restrict the number of fixed edges that may exist in succession.

Lemma 6. *Let O be a candidate orientation of a swap graph G on a feasible set S. Each cycle C in O contains at least one free edge.*

Proof: If not, then all edges of C are fixed and therefore every candidate orientation contains C. On the other hand, since S is feasible, there exists a tangle T that realizes it. But then O_T is a tangle orientation, i.e., an acyclic candidate orientation; a contradiction. □

Lemma 7. *Let G be a feasible swap graph with candidate orientation O and let C be a shortest cycle of length $m \geq 4$. Then C contains no two consecutive fixed edges.*

Proof: We first consider the case $m = 4$. Let $C = (ab, bc, cd, ad)$ be a shortest cycle such that $\{ab, bc\}$ and $\{bc, cd\}$ are fixed edges, i.e, $ac, bd \notin V$. Then all edges of the cycle are already fixed, which contradicts Lemma 6.

Now we assume $m > 4$. Thus, our cycle is of the form $C = (ab, bc, cd, de, \dots, ax)$ with $|C| \geq 5$. Further, let $bx, ac \notin V$, that is, let the edges $ax \to ab, ab \to bc$ be fixed. As we saw in the previous Lemma 6, we can assume that at least one edge is free. So let the edge $bc \to cd$ be free with $bd \in V$. By Lemma 5 no two edges can be free in succession either, which is why $cd \to de$ must be fixed with $ce \notin V$. We thus obtain the graph as in Figure 10, with the colored edges still undirected. Consider the directed edge $ab \to bc$ and, since it is fixed, we obtain either $b \prec a \prec c$ or $c \prec a \prec b$. Hence,

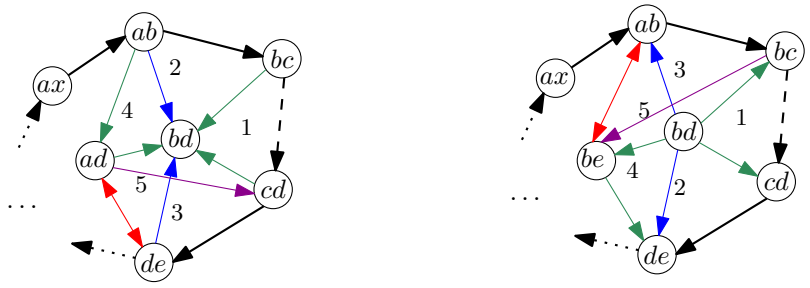


Figure 11: Here we can see the two subcases 2.a and 2.b of the proof of Lemma 7. The thick solid edges are fixed, the dashed ones are free, and the dotted ones can be either. The color scheme again corresponds to that in the proof of the Lemma 5. The reasoning for the directions is in the proof.

we assume $b \prec a \prec c$, since the other case is symmetric. With the fixed edge $ax \to ab$ we obtain $b \prec x \prec a \prec c$. For the last fixed edge $cd \to de$ we get two possibilities: $b \prec x \prec a \prec c \prec d, e \prec c$ and $b \prec x \prec a \prec c \prec e, d \prec c$, which we will treat separately.

1. Case $b \prec x \prec a \prec c \prec d, e \prec c$, compare Figure 10: Now that we know the order of the bcd-triple, we can use the direction $bc \to cd$ to infer the direction of two edges $bc \to bd$ and $bd \to cd$. To avoid the cycle (ab, bc, bd), we direct $ab \to bd$ and cannot direct the edge (bd, de) without forming a shorter cycle $(de, \dots, ax, ab, bd)$ or (bd, cd, de), which is a contradiction.

2. Case $b \prec x \prec a \prec c \prec e, d \prec c$: Since we cannot make a more precise statement here for the wire d than $d \prec c$, we further distinguish the following two cases:

   (a) Case $d \prec b$, compare Figure 11 on the left: Now that the order of the wires in the bcd-triple is known, we can use this and the orientation $bc \to cd$ to derive the direction of the triple, which is $bc \to bd$ and $cd \to bd$. The edges $ab \to bd$ and $de \to bd$ must be directed that way, since otherwise a shorter cycle either (ab, bc, bd) or $(ab, bd, de, \dots, ax)$ would arise. Further, by Lemma 4 the swap ad must be contained in V because of the order $d \prec b \prec a$ and $ab, bd \in V$ and the edges in this swap triple can be oriented. We obtain $ab \to ad$ and $ad \to bd$. Since the swap ac does not exist by assumption, the edge $ad \to cd$ is fixed. Now there exists either the (cd, de, ad)- or the $(ad, de, \dots, ax, ab)$-cycle because of the edge (de, ad), both of which are shorter than C, contradicting the assumption that the cycle C is minimal.

   (b) Case $b \prec d$, see Figure 11 on the right: again, here the bcd-triple is uniquely determined by the edge $bc \to cd$ and the order $b \prec d \prec c$, yielding $bd \to bc$ and $bd \to cd$. To avoid the (bd, cd, de)-cycle $bd \to de$ must apply, and to avoid the $(bd, de, \dots, ax, ab)$ -cycle $bd \to ab$ must also apply. The edge $bd \to de$ cannot be fixed, because $b \prec d \prec e$ holds. So the swap be must exist. Because of $bd \to de$, we direct the bde triple with $bd \to be$ and $be \to de$. Since the swap ce does not exist by assumption, the edge $bc \to be$ must be fixed. Now the edge (ab, be) always gives rise to a shorter cycle $(ab, bc, de, \dots, ax)$ or (ab, bc, be).

Thus, we see that for all orders a smaller cycle occurs, showing the statement. □

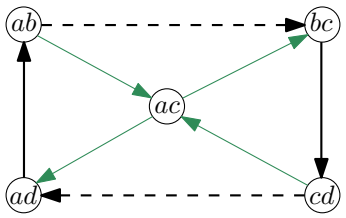


Figure 12: The graph for the proof of the Theorem 4 for a cycle of length $m = 4$. The thick edges represent the cycle, the dashed edges are free and the solid edges are fixed. The green edges originate due to the order $c \prec b \prec d \prec a$, which can be derived from the fixed edges. We immediately obtain two cycles of size 3: (ab, ac, ad) and (bc, cd, ad).

We are now ready to prove the main theorem.

Theorem 4. *Let G be a swap graph of a simple swap set S and let O be a candidate orientation of G. If O is not acyclic, then it contains a cycle of length 3.*

Proof:

Again, suppose that there is no smallest cycle of size 3 in the cyclic tangle orientation O. We first consider the case $m = 4$. So let $C = (ab, bc, cd, ad)$ be the minimal cycle, see Figure 12. By the Lemmas 7 and 5, two edges in succession can be neither both fixed nor both free. So let $ad \to ab$ and $bc \to cd$ be fixed and $ab \to bc$ and $cd \to ad$ be free. Thus, we have $ac \in V, bd \notin V$. Now consider the edge $bc \to cd$, we can conclude $c \prec b \prec d$ or $d \prec b \prec c$ since it is a fixed edge. Because of symmetry, we assume $c \prec b \prec d$. With the fixed edge $ad \to ab$ it then follows that the order $c \prec b \prec d \prec a$ applies. Now we can orient the two abc- and acd-triples with $ab \to ac \to bc$ and $cd \to ac \to ad$ and immediately obtain two smaller cycles (ab, ac, ad) and (bc, cd, ad), which contradicts the minimality of the cycle C.

Now suppose there is a minimal cycle $C := (ab, bc, cd, \dots, ax)$ of size $m > 4$. Thus, as shown in the previous lemmas, exactly every other edge is in a swap triple. Here let $ac \in V$ and thus include the abc-triple, which implies $bx, bd \notin V$. We now distinguish the following three cases for the ordering of the wires a, b and c:

1. Case $a \prec b \prec c$: Consider Figure 13 on the left. The green edges $ab \to ac$ and $ac \to bc$ follow according to condition (II) from the order $a \prec b \prec c$ and the edge $ab \to bc$. Then, to get no (ab, ac, ax)-cycle, the edge $ax \to ac$ must be contained (blue). Now the edge between ac and cd always leads to a shorter cycle (consider the cycles $(\dots, ax, ac, cd, \dots)$ and (ac, bc, cd)).

2. Case $a \prec c \prec b$: This case is shown in the middle of the Figure 13. Again, the order $a \prec c \prec b$ with the edge $ab \to bc$ already provides the directions $ac \to ab$ and $ac \to bc$ by condition (II) (green, 1). To avoid shorter cycles, the direction $ac \to cd$ must apply (otherwise (ac, bc, cd) is a smaller cycle) and then $ac \to ax$ must also apply (otherwise $(\dots, ax, ac, cd, \dots)$ is a shorter cycle; blue). The fixed edge $bc \to cd$ by condition (I) and $c \prec b$ imply the order $c \prec b \prec d$. Thus we have $a \prec c \prec b \prec d$. By Lemma 4 and $ac, cd \in V, a \prec c \prec d$ it follows that ad is also contained in the graph. Further, we can apply condition (II) to the acd-triple and obtain $ac \to ad$ and $ad \to cd$ (green, 4) because of $ac \to cd$. The edge $ab \to ad$ is directed this way because by assumption $bd \notin O$ and thus the edge has to be fixed by condition (I) and the order seen above (purple). Now we cannot orient the edge (ax, ad), otherwise we

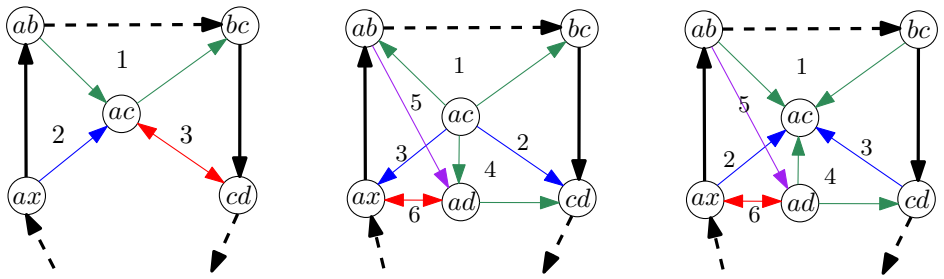


Figure 13: The three cases from left to right of the proof of Theorem 4 for $m > 4$. The orders from left to right are $a \prec b \prec c$, $a \prec c \prec b$, $b \prec a \prec c$. The color scheme is the same as in the proof of the Lemma 5 and further justification can be found in the proof.

get either the shorter cycle (ab, ad, ax) or $(\dots, ax, ad, cd, \dots)$, which forms a contradiction to the minimality of the cycle C.

3. Case $b \prec a \prec c$: See Figure 13 on the right. Condition (II) with the arrangement $b \prec a \prec c$ in the abc-triple and $ab \to bc$, yields $ab \to ac, bc \to ac$ (green, 1). To avoid a shorter cycle (ab, ac, ax) it must be $ax \to ac$ and then because of the cycle $(\dots, ax, ac, cd, \dots)$ also $cd \to ac$ (blue, 2, 3). Since $bc \to cd$ is fixed, $d \prec b \prec a \prec c$ must hold according to condition (I). Since $cd \to ac$ holds, ad must exist, otherwise the edge would have to be fixed and pointing in the other direction because of the order $d \prec a \prec c$ and condition (I). Now we can direct the acd-triple with this order and $cd \to ac$ and we get $ad \to ac$ and $ad \to cd$. Further, by condition (III) $ab \to ad$ must be fixed (purple) since bd does not exist and $d \prec b \prec a$ holds. Now we get a shorter cycle, no matter how we direct the edge between ax and ad (red): either $(\dots, ax, ad, cd, \dots)$ or (ab, ad, ax), which leads to a contradiction to the minimality of the cycle once again.

□

4.3 An Integer Linear Program

As shown in the previous section in Theorem 3, we can derive a minimal tangle from a minimal and acyclic tangle orientation O on a corresponding swap graph $G = (V, E)$. By Theorem 4, it suffices to find a candidate orientation that avoids 3-cycles. Moverover, due to condition (II) all swap triples are acyclic, i.e., a cycle of length 3 can only occur on three vertices with a common wire. We rewrite these properties in terms of linear equations to obtain an integer linear program (ILP).

For each edge $\{u, v\} \in E$ we introduce two binary variables $d_{(u,v)}, d_{(v,u)} \in \{0, 1\}$ that indicate that $\{u, v\}$ is directed from u to v or vice versa. Moreover, for each vertex $v \in V$ we have a variable $h_v \in \mathbb{R}$ that shall encode its height and a single variable $h \in \mathbb{R}$ that shall encode the height of the orientation. We use the reals instead of integers for the heights, since it is usually easier for a solver to use continuous values and for the solution it makes no difference. The ILP is as follows.

$$\text{minimize } h \tag{1}$$

subject to

$$d_{(u,v)} + d_{(v,u)} = 1 \text{ for all } \{u,v\} \in E \tag{2}$$
$$d_{(u,v)} = 1 \text{ for all fixed edges } \{u,v\} \in E \tag{3}$$
$$d_{(ab,ac)} = d_{(ab,bc)} = d_{(ac,bc)} \text{ for all } \{ab,ac,bc\} \subseteq V \text{ with } a \prec b \prec c \tag{4}$$
$$1 \leq d_{(ab,ac)} + d_{(ac,ad)} + d_{(ad,ab)} \leq 2 \text{ for all } \{ab,ac,ad\} \subseteq V \text{ with } a \prec b \prec c \tag{5}$$
$$0 \leq h_v \leq h \text{ for all } v \in V \tag{6}$$
$$h_v \geq h_u + 1 + |E| \cdot (d_{(u,v)} - 1) \text{ for all pairs } (u,v) \in V \times V \text{ with } \{u,v\} \in E \tag{7}$$

Observe that equation (2) ensures that the variables $d_{(u,v)}$ indeed determine an orientation O of the swap graph. Equations (3) and (4) enforce conditions (I) and (II). Equation (5) forbids 3-cycles in the orientation and by Theorem 4 therefore implies that O is acyclic, i.e., it is a tangle orientation. Equations (6) and (7) ensure that for each edge the height of the target is at least the height of the source plus 1 and that h is an upper bound for all height values of all vertices. Note that equation (7) is trivially satisfied for a pair (u,v) if $d_{(u,v)} = 0$. Further, the fact that h is minimized, ensures that h is the height of the orientation O. We conclude that the optimal solution of the ILP are precisely the acyclic tangle orientations of minimum height.

5 Experimental Evaluation

In this section we experimentally evaluate the performance of the two optimal solvers for the simple tangle height minimization problem and compare it with the algorithm proposed by Firman et al. [5, 6]. As each of our algorithms offers several degrees of freedom and allows several optimizations, we first compare them individually and determine the best-performing versions of the search-based algorithm from Section 3 and of the ILP-based approach from Section 4 that are based on the notion of swap graphs. Finally, we compare the best variants with each other on somewhat larger instances.

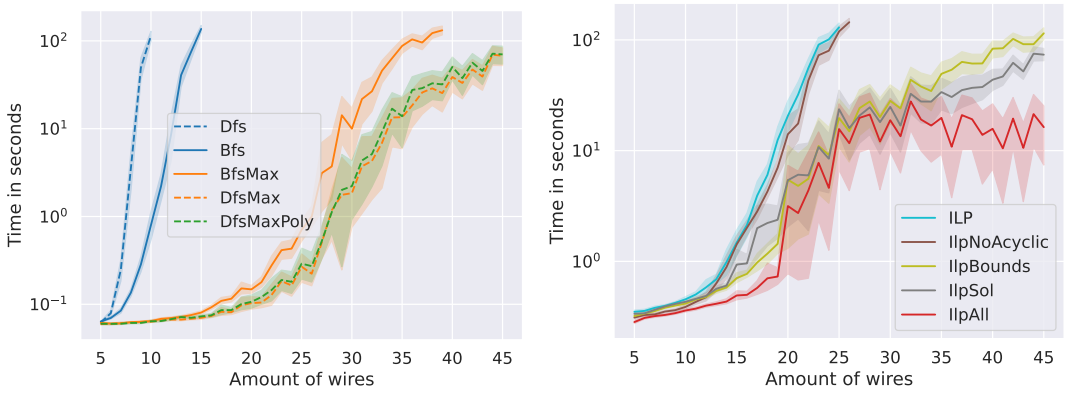
As test data for the initial comparison of the different variants of each approach we chose the natural ordering $1, 2, \ldots, n$ as the start ordering and generated random final orderings of $\{1, \ldots, n\}$ for $n = 5, \ldots, 45$. For each size n we generated 100 random instances.

All experiments were run on a single core of an Intel Xeon E5-2620v4 @ 2.10 GHz Processor, with 8GiB of RAM directly connected, running Debian 11 (bullseye) on kernel version 5.10.0-13. All algorithms are implemented in Java and run on an OpenJDK Runtime Environment (build 11.0.14). We use gurobi[2] version 9.5.0 as ILP solver. We use a time limit of 180s per instance in our experiments. If an algorithm runs out of memory (OOM), we treat the same as if it would have exceeded the time limit (TLE).

5.1 Comparison of Search-Based Approaches

In this section we compare the running times of several algorithms that are based on the ideas behind Theorem 2. We refer to the original algorithm by Firman et al. [5,6], which performs a BFS

[2] www.gurobi.com



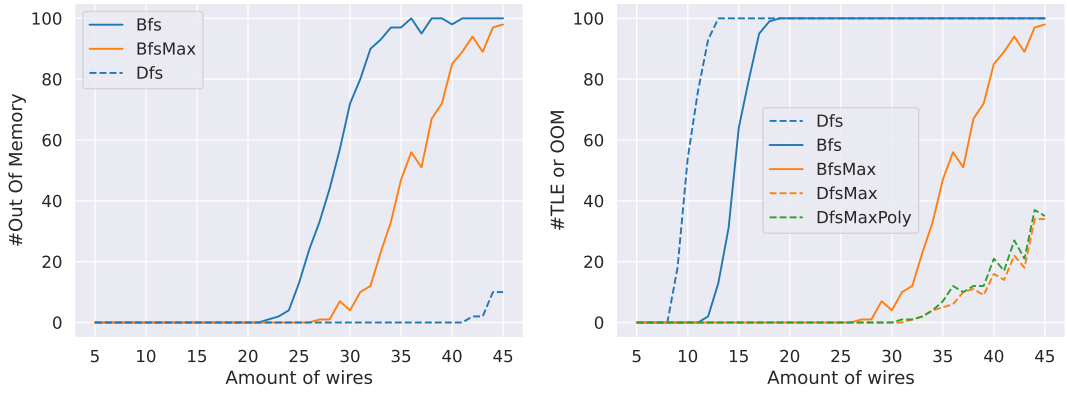
(a) (b)

Figure 14: Comparison of the running times of the search-based algorithms (a) and the ILP approaches (b). The time limit was set to 180 seconds and sizes where less than 75% of the instances could be solved are cut off. The shaded areas indicate the standard deviation.

on the graph of permutations as `Bfs`. To reduce the memory footprint in practice, we construct the graph of permutations on the fly rather than constructing it up-front and then searching a shortest path. Based on Theorem 1, we can limit our exploration of this graph along edges that correspond to maximal swap sets. We refer to this variant of the algorithm as `BfsMax`.

An alternative to this is a search based on a DFS. To limit the search depth, we use Wang's algorithm to compute a solution that has height $h \leq \mathrm{OPT} + 1$. We then limit the search depth of the DFS to $h - 1$. If we find a solution at depth $h - 1$, it must be optimal. Otherwise, we know that the solution computed by Wang's algorithm is optimal. Following the naming scheme for the BFS-based algorithms, we denote the basic version of this approach by `Dfs`. The variant that explores only edges that correspond to maximal swap sets is called `DfsMax`, while the variant that uses only $O(n^2)$ space by enumerating the outgoing edges one by one with amortized constant delay is called `DfsMaxPoly`.

Figure 14a shows the running times with respect to the number n of wires. Both `Bfs` and `Dfs` consistently hit the time limit of 180s below 20 wires. The restriction to maximal swap sets considerably improves the performance and lets them solve instances that are far larger. In terms of running time, `DfsMaxPoly` is en par with `DfsMax`, indicating that the overhead incurred by the enumeration is not significant. Figure 15a shows that all BFS-based algorithms are inherently memory bound. `Bfs` hits the memory bound on nearly all instances starting from about 35 wires. However, as Figure 15b shows, even though memory is not yet an issue, it consistently exceeds the time limit even below 20 wires. By comparison `Dfs` very rarely runs out of memory, even for $n \geq 40$ but it exceedingly slows and does not even manage to solve any instance with 15 wires. The restriction to maximal swap sets greatly improves the performance, even more so for `BfsMax` than for `DfsMax`. However starting from about 24 wires, `BfsMax` starts to run out of memory for about 35% of the instances in total. By constrast, `DfsMax` and `DfsMaxPoly` never run out of memory and exceed the time limit for only few instances in total. Overall, `BfsMax`, `DfsMax` and `DfsMaxPoly` are the strongest candidates that perform equally well. The former tends to run out of memory from 40 wires onwards, but if it goes through, it is usually slightly faster than the latter two. Since `DfsMax` and `DfsMaxPoly` seem to perform equally well and there is no big



(a) (b)

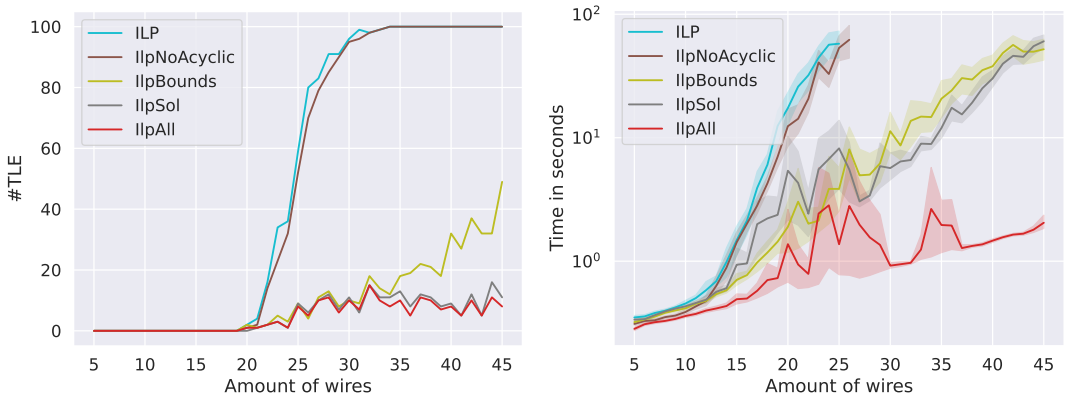
Figure 15: The number of instances for which the algorithms went out of memory (a) or exceeded the time limit (b).

algorithmic difference, we only compare the `DfsMaxPoly` in our large comparison to the rest.

5.2 The Integer Linear Program

In this section, we experimentally evaluate the ILP from Section 4, which is based on the swap graph. We consider several variants of it. First, the basic ILP as described in Section 4.3, which we denote by `ILP`. Second, it is possible to provide the solver also with a solution to the ILP and we use the one computed by Wang's algorithm. We call this approach `IlpSol`. Third, it can be seen that if the orientation encoded by the $d_{(u,v)}$ variables of the `ILP` is not acyclic, then there is no feasible value for h. It follows that the ILP obtained by dropping Equation (5) is equivalent to the original ILP. We refer to this as `IlpNoAcyclic`. Fourth, using Wang's algorithm, we can compute a solution of height $h' \leq \mathrm{OPT}+1$. We can then add the constraints $h \geq h'-1$ and $h \leq h'$ to limit the search space of the ILP. The reason for this is that often ILP-based approaches spend a significant amount of time on proving the optimality of a solution candidate. The bounds in this case allow the ILP solver to immediately conclude that a solution of height $h'-1$ is optimal and in case it disproves the existence of a solution of height $h'-1$, it can immediately conclude that height h' is optimal. We call the variant `ILP` with these bounds `IlpBounds`. Finally, we also consider a variant `IlpAll` that combines all of these optimizations.

In each implementation, given an input ordering, we first compute its swap graph and determine the complete triples and the fixed edges. Afterwards we construct the corresponding ILP and solve it using gurobi. Figure 16a shows the running time of the ILP-based approaches on the test instances. The evaluation shows that starting from size 20 all ILPs start running into timeouts for some of the instances. Dropping the acyclicity constraints barely improves the performance over the baseline ILP, and `ILP` and `IlpNoAcyclic` barely solve instances above size 30. Surprisingly `IlpSol` performs far better, and it even beats `IlpBounds`, which indicates that even finding a feasible solution for the ILP can be far from trivial. `IlpAll` overall performs best and solves about 90% of the instances even for 45 wires within the time limit, beating `IlpSol` in terms of running time for larger instances. Overall we can conclude that `IlpAll` is clearly superior to all other variants. Interestingly, Figure 16b shows that, after filtering the difficult instances that hit



(a) (b)

Figure 16: Performance comparison of the different ILP approaches with time limit 180s. In (a) we count the amount of instances that exceeded the time limit, in (b) they are filtered out and the graph is cut off, if there more than 75% of the instances fail.

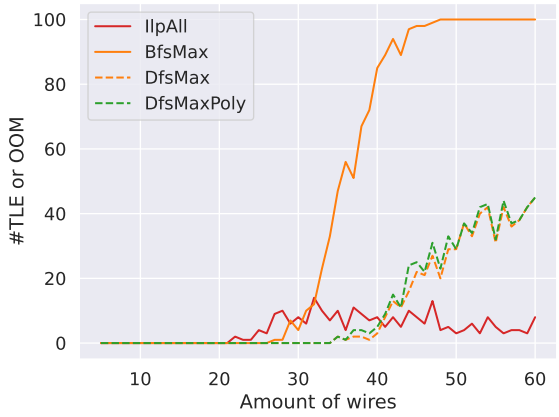
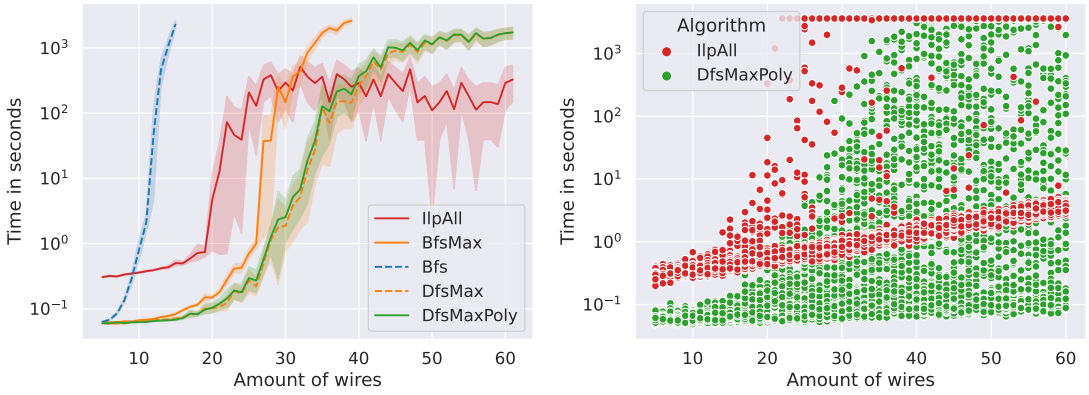


Figure 17: Timeouts and out of memory for the best variants of each approach.



(a) (b)

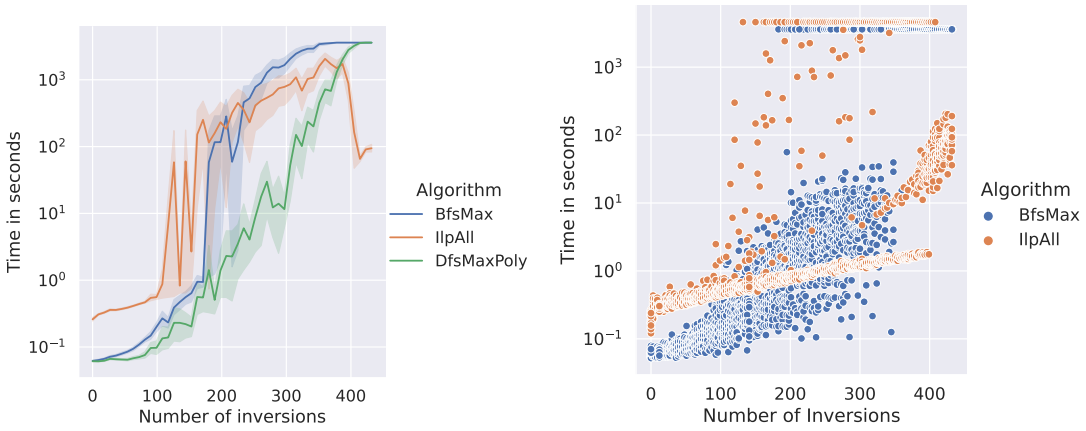
Figure 18: (a) Runnimg times of the best search-based algorithms and the best ILP. (b) A scatter plot showing the running times for `IlpAll` and `DfsMaxPoly`. Every point corresponds to one instance. The time limit was one hour, and out of memory errors were interpreted as time limit exceeded and their time increased accordingly.

the time limit, all ILP variants exhibit a clear exponential dependency on the ordering size n.

5.3 Overall Comparison

We now compare the most efficient variants of each of the different approaches and BFS (for comparison with Firman et al.), i.e., `BfsMax`, `DfsMaxPoly`, and `IlpAll` on larger instances of size up to 60 and with a time limit of $1h$. As before, we generate 100 random orderings for each size.

Figure 18a shows the running times. We see, that `Bfs` cannot handle instances of size 20 or more. For ordering sizes up to 40 the new search-based algorithms are superior to `IlpAll`. However, for larger instances `IlpAll` preserves good performance. On the other hand, `BfsMax` consistently runs out of memory from size 40, while the running time of `DfsMax` and `DfsMaxPoly` become increasingly worse. The Figure also suggests that `DfsMax` and `DfsMaxPoly` behave basically the same on those small instances. This is no surprise as we basically need the same time for computing the outgoing edges on demand or all at once. Also the improvement to polynomial space seems unimportant for $n \leq 60$, as `DfsMax` does not run out of space. A more detailed look at the running times for `IlpAll` and `DfsMaxPoly` (for clarity we omit `DfsMax`, as it looks basically the same) in Figure 18b reveals that the search-based algorithms are consistently faster than `IlpAll` up to size 25. From then on, their performance starts to vary wildly, which is likely explained by the point of time when the algorithm finds a better solution than the one given by Wang's algorithm. `IlpAll`, on the other hand, exhibits a clear but moderate exponential dependency of its running time on the number of wires, which in Figure 18a is somewhat obscured by the relatively few instances (5–10%) where `IlpAll` exceeds the time limit. A detailed look at the number of instances where the algorithms fail due to running out of memory or exceeding the time limit (see Figure 17) reveals that `IlpAll` does not solve about 10% of the instances up to size 60, whereas `DfsMax` and `DfsMaxPoly` fail on more than 20% of the instances of size 45, and this increases to about 50% for size 60. Our recommendation is therefore to exclusively use `DfsMaxPoly` up to size 30 and to run `IlpAll` and `DfsMaxPoly` in parallel for larger instances.



(a) (b)

Figure 19: The amount of wires is $n = 30$ everywhere, and agian out of memory errors where interpreted as time limit exceeded and their needed time raised accordingly. (a) Running time with respect to the number of inversions in the final ordering for `BfsMax`, `IlpAll`, and `DfsMaxPoly`. To smoothen the plot, the swap sizes are grouped into buckets of size 9.(b) Scatter plot of the running times of `BfsMax` and `IlpAll`. The points that correspond to instances where `IlpAll` exceeds the time limit have been slightly displaced to remove the overlap with the instances where `BfsMax` exceeds the time limit or runs out of memory.

It is an interesting question whether the difficulty of an instance is correlated with the number of swaps, i.e., the number of inversions of the final ordering. To answer this question, we generated an additional data set with a fixed the number of $n = 30$ wires that contains final orderings with k inversions for $k \in \{0, 1, 2, \ldots, 435\}$. We generated 7 instances for each number of inversions. The results of the algorithms are shown in Figure 19a. For `DfsMaxPoly` there is a clear correlation between the number of inversions and the running time. For `BfsMax` and `IlpAll` such a correlation is also present, but the results seem to vary more strongly depending on the actual input. The scatter plot in Figure 19b reveals that there are difficult instances for these algorithms even with relatively few inversions and generally the running time increases with the number of inversions. As the number of inversions approach the maximum, `BfsMax` starts to consistently run out of memory. On the other hand, `IlpAll` also works well for instances whose number of inversions is close to the maximum, which intuitively makes sense since lower bounds for such instances are most likely easier to obtain.

6 Conclusion

We have conducted an experimental study of height-minimization algorithms for simple tangles. First, we have developed search-based exponential-time algorithms that improve over the previous approach by Firman et al. [5, 6]. Second, we have transformed the tangle height minimization problem for simple tangles into a graph orientation problem, which allowed us to formulate it as an ILP. Our experimental evaluation shows that our algorithms can solve significantly larger instances than previous algorithms. We hope that the ability to solve larger instances and the connection to

graph orientation problems may be useful tools for addressing our main open question: Can the height of simple tangles be computed in polynomial time?

References

[1] V. Alistarov. Computing tangles using a SAT solver. Course report, University of Würzburg, 2022. https://www1.pub.informatik.uni-wuerzburg.de/pub/theses/2022-alistarov-masterpraktikum.pdf.

[2] J. Baumann. Höhenminimierung einfacher Tangles. Bachelor thesis, University of Passau, 2020. https://www.fim.uni-passau.de/fileadmin/dokumente/fakultaeten/fim/lehrstuhl/rutter/abschlussarbeiten/2020-Jakob_Baumann-BA.pdf.

[3] J. Birman and R. Williams. Knotted periodic orbits in dynamical systems—I: Lorenz's equation. *Topology*, 22(1):47–82, 1983. doi:10.1016/0040-9383(83)90045-9.

[4] O. Firman, P. Kindermann, B. Klemz, A. Ravsky, A. Wolff, and J. Zink. The complexity of finding tangles. In L. Gasieniec, editor, *Proceedings of the 48th International Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM'23)*, volume 13878 of *Lecture Notes in Computer Science*, pages 3–17. Springer, 2023. doi:10.1007/978-3-031-23101-8_1.

[5] O. Firman, P. Kindermann, B. Klemz, A. Ravsky, A. Wolff, and J. Zink. Deciding the feasibility and minimizing the height of tangles. *J. Graph Algorithms Appl.*, 29(1):135–158, 2025. doi:10.7155/JGAA.V29I1.3053.

[6] O. Firman, P. Kindermann, A. Ravsky, A. Wolff, and J. Zink. Computing height-optimal tangles faster. In D. Archambault and C. D. Tóth, editors, *Proceedings of the 27th International Symposium on Graph Drawing and Network Visualization (GD'19)*, volume 11904 of *Lecture Notes in Computer Science*, pages 203–215. Springer, 2019. doi:10.1007/978-3-030-35802-0_16.

[7] M. C. Golumbic. *Algorithmic Graph Theory and Perfect Graphs*, chapter Permutation Graphs, pages 157–170. Elsevier, 1980.

[8] D. E. Knuth. *The Art of Computer Programming. Volume 4, Fascicle 2 - Generating All Tuples and Permutations*. Addison-Wesley, 2005.

[9] D. E. Knuth. *The Art of Computer Programming. Volume 4, Fascicle 3 - Generating All Combinations and Partitions*. Addison-Wesley, 2005.

[10] G. Mindlin, X. Hou, R. Gilmore, H. Solari, and N. Tufillaro. Classification of strange attractors by integers. *Phys. Rev. Lett.*, 64:2350–2353, 1990. doi:10.1103/PhysRevLett.64.2350.

[11] M. Olszewski, J. Meder, E. Kieffer, R. Bleuse, M. Rosalie, G. Danoy, and P. Bouvry. Visualizing the template of a chaotic attractor. In T. Biedl and A. Kerren, editors, *Proceedings of the 26th International Symposium on Graph Drawing and Network Visualization (GD'18)*, volume 11282 of *Lecture Notes in Computer Science*, pages 106–119. Springer, 2018. doi:10.1007/978-3-030-04414-5_8.

[12] K. Sado and Y. Igarashi. A function for evaluating the computing time of a bubbling system. *Theoretical Computer Science*, 54(2):315–324, 1987. doi:10.1016/0304-3975(87)90136-8.

[13] D. C. Wang. Novel routing schemes for IC layout, part I: two-layer channel routing. In A. R. Newton, editor, *Proceedings of the 28th Design Automation Conference*, pages 49–53. ACM, 1991. doi:10.1145/127601.127626.

[14] K. Yamanaka, T. Horiyama, T. Uno, and K. Wasa. Ladder-lottery realization. In *Proceedings of the 30th Canadian Conference on Computational Geometry (CCCG'18)*, pages 61–67, 2018. URL: https://www.academia.edu/download/88658268/proceedings.pdf#page=71.