Journal of Graph Algorithms and Applications
http://jgaa.info/ vol. 29, no. 1, pp. 135–158 (2025)
DOI: 10.7155/jgaa.v29i1.3053

Deciding the Feasibility and Minimizing the Height of Tangles[*]

Oksana Firman[1] Philipp Kindermann[2] Boris Klemz[1] Alexander Ravsky[3]
Alexander Wolff[1] Johannes Zink[1]

[1]Universität Würzburg, Würzburg, Germany
[2]Universität Trier, Trier, Germany
[3]Pidstryhach Institute for Applied Problems of Mechanics and Mathematics, National Academy of Sciences of Ukraine, Lviv, Ukraine

Submitted: December 2023 Accepted: June 2025 Published: June 2025
Article type: Regular paper Communicated by: Ignaz Rutter

Abstract. We study the following combinatorial problem. Given a set of n y-monotone Jordan curves, called *wires*, a *tangle* determines the order of the wires on a number of horizontal *layers* such that the orders of the wires on any two consecutive layers differ only in swaps of neighboring wires. Given a multiset L of *swaps* (that is, unordered pairs of wires) and an initial order of the wires, a tangle *realizes* L if each pair of wires changes its order exactly as many times as specified by L. List-Feasibility is the problem of finding a tangle that realizes a given list L for a prescribed initial order if such a tangle exists. The Tangle-Height Minimization problem additionally aims to find a tangle that uses the minimum number of layers. List-Feasibility (and therefore Tangle-Height Minimization) is NP-hard [Yamanaka et al., CCCG 2018].

We prove that List-Feasibility remains NP-hard if every pair of wires swaps only a constant number of times. On the positive side, we present an algorithm for Tangle-Height Minimization that computes an optimal tangle for n wires and a given list L of swaps in $\mathcal{O}((|L|/n^2 + 1)^{n^2/2} \cdot \varphi^n \cdot n! \cdot n \cdot \min\{|L|, n^2\} \cdot \log |L|)$ time, where $\varphi \approx 1.618$ is the golden ratio and $|L|$ is the total number of swaps in L.

Obviously, this algorithm solves List-Feasibility, too. We show that List-Feasibility can also be solved by a simpler and faster version of the algorithm. Moreover, the algorithm helps us to show that List-Feasibility is in NP and fixed-parameter tractable with respect to the number of wires. For *simple* lists, where every swap occurs at most once, we show how to solve Tangle-Height Minimization in $\mathcal{O}(n!\varphi^n)$ time.

[*]Preliminary versions of this work have appeared in Proc. 27th Int. Symp. Graph Drawing & Network Vis. (GD 2019) [8] and Proc. 49th Int. Conf. Current Trends Theory & Practice Comput. Sci. (SOFSEM 2023) [7]

E-mail addresses: oksana.firman@uni-wuerzburg.de (Oksana Firman) kindermann@uni-trier.de (Philipp Kindermann) boris.klemz@uni-wuerzburg.de (Boris Klemz) vitaravski@gmail.com (Alexander Ravsky) johannes.zink@uni-wuerzburg.de (Johannes Zink)

1 Introduction

This paper concerns the visualization of *chaotic attractors*, which occur in dynamic systems. Such systems are considered in physics, celestial mechanics, electronics, fractals theory, chemistry, biology, genetics, and population dynamics; see, for instance, [6], [17], and [10, p. 191].

The literature contains many definitions of an attractor, so here we just give some intuition. Namely, an attractor can be understood as a minimal set A of states towards which a system evolves when its state belongs to some neighborhood of A in the space of the system states [20]. Simple examples of attractors can be fixed points or limit cycles. However, there are also chaotic attractors. They exhibit a sensitive dependence on initial conditions (known as the "butterfly effect"), when arbitrarily small changes in an initial system state can result in its essentially different evolution, in particular, in large differences in a later state. Chaotic attractors can have a very complicated (even fractal) structure, so it can be difficult to describe them. Nevertheless, in some cases, we can describe them at least roughly. Studying the topological structure of chaotic attractors, Birman and Williams [5] investigated how they are knotted. Later Mindlin, Hou, Gilmore, Solari, and Tufillaro [13] described attractors using so-called *linking matrices*. Such a matrix contains the number of torsions (on the main diagonal) and the number of oriented crossings (off the diagonal) that occur among the attractor parts.

Olszewski, Meder, Kieffer, Bleuse, Rosalie, Danoy, and Bouvry [14] studied the problem of visualizing the topological structure of chaotic attractors. They focused on the combinatorial part of the problem, thus ignoring torsions and the orientations of the crossings. In the framework of their paper, one is given a set of n y-monotone Jordan curves called *wires* that hang off a horizontal line in a fixed order, and a multiset of swaps between the wires; a tangle then is a visualization of these swaps, i.e., a sequence of horizontal layers with an order of the wires on each layer such that (i) only adjacent wires are swapped and (ii) only disjoint swaps can be done simultaneously. For examples of lists of swaps (described by a multilist and by an $(n \times n)$-matrix) and tangles that realize these lists, see Figs. 1 and 2.

Olszewski et al. gave an exponential-time algorithm for minimizing the *height* of a tangle, that is, the number of layers. We call this problem TANGLE-HEIGHT MINIMIZATION. They tested their algorithm on a benchmark set.

In an independent line of research, Yamanaka, Horiyama, Uno, and Wasa [21] showed that the problem LADDER-LOTTERY REALIZATION is NP-hard. As it turns out, this problem is equivalent to deciding the feasibility of a list, i.e., deciding whether there exists a tangle realizing the list for a prescribed initial order. We call this problem LIST-FEASIBILITY.

Sado and Igarashi [15] used the same objective function for *simple* lists, that is, lists where each swap appears at most once. (In their setting, instead of a list, the start and final permutations are given but this uniquely defines a simple list of swaps.) They used odd-even sort, a parallel variant of bubblesort, to compute tangles with at most one layer more than the minimum. Their algorithm runs in quadratic time. Wang [19] showed that there is always a height-optimal tangle where no swap occurs more than once. Bereg, Holroyd, Nachmanson, and Pupyrev [3,4] considered a similar problem. Given a final permutation, they showed how to minimize the number of bends or *moves* (which are maximal "diagonal" segments of the wires).

Let $L^1 = (l^1_{ij})$ denote the (simple) list with $l^1_{ij} = 1$ if $i \neq j$, and $l^1_{ij} = 0$ otherwise. This list is feasible even if we start with any permutation of $\{1, \ldots, n\}$; a tangle realizing L^1 is commonly known as *pseudo-line arrangement*. So tangles can be thought of as generalizations of pseudo-line arrangements where the numbers of swaps are prescribed and even feasibility becomes a difficult question.

In this paper we give new, faster algorithms for List-Feasibility and Tangle-Height Minimization, but before we can present our contribution in detail, we need some notation.

Framework, Terminology, and Notation. A *permutation* is a bijection of the set $[n] = \{1, \dots, n\}$ onto itself. The set S_n of all permutations of the set $[n]$ is a group whose multiplication is a composition of maps (i.e., $(\pi\sigma)(i) = \pi(\sigma(i))$ for each pair of permutations $\pi, \sigma \in S_n$ and each $i \in [n]$). The identity of the group S_n is the identity permutation $\mathrm{id}_n = \langle 1, 2, \dots, n\rangle$. We write a permutation $\pi \in S_n$ as the sequence of numbers $\langle \pi^{-1}(1), \pi^{-1}(2), \dots, \pi^{-1}(n)\rangle$. In this sequence, element $i \in [n]$ is placed at the position $\pi(i)$. For instance, the permutation $\pi \in S_4$ with $\pi(1) = 3$, $\pi(2) = 4$, $\pi(3) = 2$, and $\pi(4) = 1$ is written as $\langle 4, 3, 1, 2\rangle$ (or simply as 4312).

Two permutations π and σ of S_n are *adjacent* if they differ only in transposing neighboring elements, that is, if, for every $i \in [n]$, $|\pi(i) - \sigma(i)| \le 1$. Then the set $\{i \in [n] : \pi(i) \ne \sigma(i)\}$ splits into pairs $\{i, j\}$ such that $\pi(i) = \sigma(j)$ and $\pi(j) = \sigma(i)$. Each such pair is called a *swap*. For two adjacent permutations π and σ, let

$$\mathrm{diff}(\pi, \sigma) = \big\{(i, j) \mid i, j \in [n] \ \wedge\ i \ne j \ \wedge\ \pi(i) = \sigma(j) \ \wedge\ \pi(j) = \sigma(i)\big\}$$

be the set of swaps in which π and σ differ. Given a permutation π and a pair $(i, j) \in [n]^2$ with $|\pi(i) - \pi(j)| = 1$, *applying* the swap (i, j) to π yields an adjacent permutation σ such that $\pi(i) = \sigma(j)$ and $\pi(j) = \sigma(i)$. Given a set of y-monotone Jordan curves called *wires* that hang off a horizontal line, we label them by their index in a prescribed start permutation π_1 (which we will assume to be id_n). Furthermore, we define a *list* $L = (l_{ij})$ of *order* n to be a symmetric $n \times n$ matrix with entries in $\mathbb{N}_0$ and zero diagonal. A list $L = (l_{ij})$ can also be considered as a multiset of swaps, where l_{ij} is the multiplicity of swap (i, j). By $(i, j) \in L$ we mean $l_{ij} > 0$.

A *tangle* T of *height* h realizing L is a sequence $\langle \pi_1, \pi_2, \dots, \pi_h\rangle$ of permutations of the labels of the wires such that (i) consecutive permutations are adjacent and (ii) $L = \bigcup_{i=1}^{h-1} \mathrm{diff}(\pi_i, \pi_{i+1})$. (Recall that L is a multiset, so the union in (ii) can yield several copies of the same swap.) A *subtangle* of T is a non-empty sequence $\langle \pi_p, \pi_{p+1}, \dots, \pi_q\rangle$ of consecutive permutations of T (that is, $1 \le p \le q \le h$).

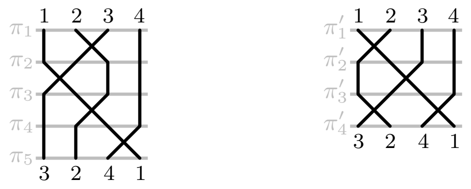


Figure 1: Tangles T and T' of different heights realizing the list $L = \{(1, 2), (1, 3), (1, 4), (2, 3)\}$.

In this paper, we assume that the start permutation π_1 of a tangle is always id_n (except if explicitly stated otherwise). For example, the list L in Fig. 1 admits a tangle that starts with id_4 and realizes L. We call such a list *feasible*. For example, the list $L' = L \cup \{(1, 2)\}$ with two $(1, 2)$ swaps, in contrast, is not feasible (assuming that the start permutation is id_4).

In Fig. 2, the list L_n is feasible; it is specified by an $(n \times n)$-matrix. The gray horizontal bars correspond to the permutations (or *layers*). As a warm-up, we characterize the tangles that realize L_n; this characterization will be useful in Section 2.

Observation 1 *Given the start permutation* id_n*, the tangle in Fig. 2 realizes the list* L_n *specified there. All tangles that start with* id_n *and realize* L_n *have the same order of swaps along each wire.*

$$L_n = \begin{pmatrix} 0 & 1 & 1 & \dots & 1 & \mathbf{0} & \mathbf{2} \\ 1 & 0 & 1 & \dots & 1 & \mathbf{2} & \mathbf{0} \\ 1 & 1 & 0 & \dots & 1 & \mathbf{0} & \mathbf{2} \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 1 & 1 & 1 & \dots & 0 & 0 & 2 \\ \mathbf{0} & \mathbf{2} & \mathbf{0} & \dots & 0 & 0 & n-1 \\ \mathbf{2} & \mathbf{0} & \mathbf{2} & \dots & 2 & n-1 & 0 \end{pmatrix}$$

(The bold zeros and twos must be swapped if n is even.)

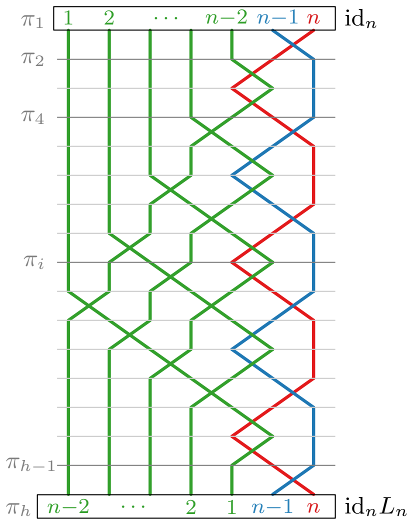


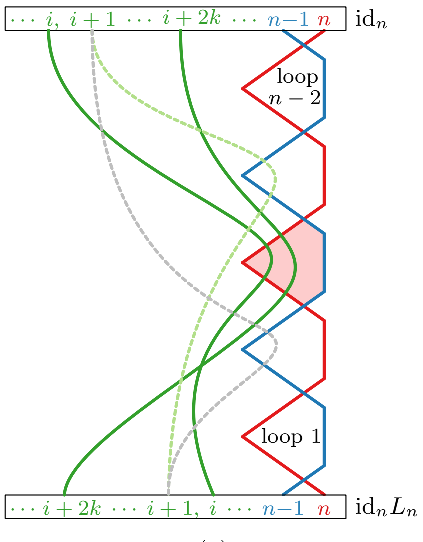
Figure 2: A list L_n for n wires and a tangle of height $3n-4$ realizing L_n (drawn for $n = 7$). The tangle height is minimum.

Proof: We call wires $n-1$ and n *loop wires*. We call each region that is formed by these two wires between two consecutive swaps a *loop*. We number the loops from bottom to top $1, 2, \dots, n-2$; see Fig. 3a. Clearly, for every $i \in [n-2]$, wire i enters exactly one of these loops (of the right parity) from the left and leaves it to the left (because it swaps only with one of the two loop wires). We claim that wire i enters loop i. Assuming that this is true, before wire i can enter its loop, it must have swapped with wires $i+1, i+2, \dots, n-2$ and it cannot have swapped with any of the wires $1, 2, \dots, i-1$ to reach its loop. By induction over i, the order of the wires that swap with wire i is (going from top to bottom) $n-2, n-3, \dots, i+1$, then $n-1$ or n (depending on the parity of i), and then $i-1, i-2, \dots, 1$.

To see our claim, observe the following. If two wires (of the same parity), say i and $i' = i+2k$ (for some $k > 0$), enter the same loop, then wire $i+1$ (which starts between wires i and i', but has different parity) has to swap at least twice with wire i or with wire i' in order to reach a loop of the right parity and then its final destination between wires i' and i; see the two dashed curves in Fig. 3a. Hence, each loop is entered by exactly one non-loop wire. Assume that not every wire enters "its" loop. Let i be the wire with the largest index that enters a loop $l > i$. Then there is a wire $i' > i$ of the same parity as i that enters a loop $l' < l$; see Fig. 3b. Then, however, wire i' would have to swap at least twice with wire i. □

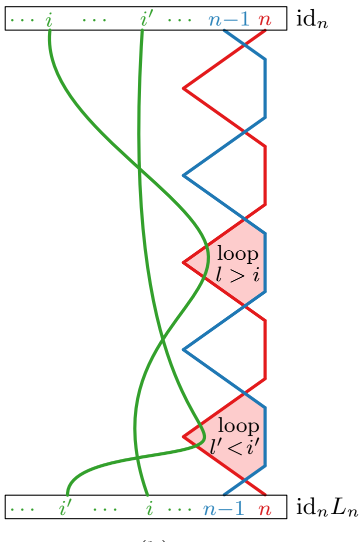
Let $|L| = \sum_{i<j} l_{ij}$ be the *length* of L. A list $L' = (l'_{ij})$ is a *sublist* of L if $l'_{ij} \le l_{ij}$ for each $i, j \in [n]$. If there is a pair $(i', j') \in [n]^2$ such that $l'_{i'j'} < l_{i'j'}$, then L' is a *strict* sublist of L. A list is *simple* if all its entries are zeros or ones. A list $L = (l_{ij})$ is *even* if all l_{ij} are even, and *odd* if all non-zero l_{ij} are odd. For any two lists $L = (l_{ij})$ and $L' = (l'_{ij})$ such that, for each $i, j \in [n]$, $l'_{ij} \le l_{ij}$, let $L - L' = (l_{ij} - l'_{ij})$.

In order to understand the structure of feasible lists better, we consider the following relation between them. Let $L = (l_{ij})$ be a feasible list. Pick wires i' and j' with $l_{i'j'} > 0$ and let $L' = (l'_{ij})$ be the list with $l'_{ij} = l_{ij}$ for every $(i, j) \in [n]^2 \setminus \{(i', j'), (j', i')\}$ and $l'_{i'j'} = l'_{j'i'} = l_{i'j'} + 2 (= l_{j'i'} + 2)$. We claim that the list L' is feasible, too. To this end, note that any tangle T that realizes L has



$\cdots$ i, $i+1$ $\cdots$ $i+2k$ $\cdots$ $n-1$ n
id_n
loop $n-2$
loop 1
$\cdots$ $i+2k$ $\cdots$ $i+1$, i $\cdots$ $n-1$ n
$\mathrm{id}_n L_n$

(a)



$\cdots$ i $\cdots$ i' $\cdots$ $n-1$ n
id_n
loop $l > i$
loop $l' < i'$
$\cdots$ i' $\cdots$ i $\cdots$ $n-1$ n
$\mathrm{id}_n L_n$

(b)

Figure 3: Proof of Section 1.

two neighboring layers with adjacent permutations π and π' such that $(i,j) \in \mathrm{diff}(\pi,\pi')$. Directly after π, we can insert two (i,j) swaps into T. This yields a tangle that realizes L'. Given two lists $L = (l_{ij})$ and $L' = (l'_{ij})$, we write $L \to L'$ if $L = L'$ or if the list L can be *extended* to the list L' by iteratively applying the above operation.

For a list $L = (l_{ij})$, let $1(L) = (l_{ij} \bmod 2)$ and let $2(L) = (l''_{ij})$ with $l''_{ij} = 0$ if $l_{ij} = 0$, $l''_{ij} = 1$ if l_{ij} is odd, and $l''_{ij} = 2$ otherwise. We call $2(L)$ the *type* of L. Clearly, given two lists $L = (l_{ij})$ and $L' = (l'_{ij})$, we have that $L \to L'$ if and only if $2(L) = 2(L')$ and $l_{ij} \le l'_{ij}$ for each $i, j \in [n]$.

A feasible list L_0 is *minimal* if there exists no feasible list $L^\star \neq L_0$ such that $L^\star \to L_0$. Thus a list L is feasible if and only if there exists a minimal feasible list L_0 of type $2(L)$ such that $L_0 \to L$.

Our Contribution. As mentioned above, Yamanaka et al. [21] showed that List-Feasibility in general is NP-hard (which means that Tangle-Height Minimization is NP-hard as well). However, in their reduction, for some swaps the number of occurrences is linear in the number of wires. We strengthen their result by showing that List-Feasibility is NP-hard even if all swaps have constant multiplicity; see Section 2. Our reduction uses a variant of Not-All-Equal 3-SAT (whereas Yamanaka et al. used 1-in-3 3SAT). Moreover we show that for some classes of lists, the problem is efficiently solvable; see Section 4.

For Tangle-Height Minimization of simple lists for n wires, we present an exact algorithm that is based on breadth-first search (BFS) in an auxiliary graph and runs in $\mathcal{O}(n!\varphi^n)$ time, where $\varphi = (\sqrt{5}+1)/2 \approx 1.618$ is the golden ratio. Recently, Baumann [2] has shown that the BFS can be executed on a smaller auxiliary graph, which leads to a runtime of $\mathcal{O}(n!\psi^n)$ time, where $\psi = (\sqrt[3]{9-\sqrt{69}} + \sqrt[3]{9+\sqrt{69}})/\sqrt[3]{18} \approx 1.325$. For general lists, we present an exact algorithm that is based on dynamic programming and runs in $\mathcal{O}((|L|/n^2+1)^{n^2/2} \cdot \varphi^n \cdot n! \cdot n \cdot \min\{|L|, n^2\} \cdot \log|L|)$ time, where L is the input list; see Section 3. Note that the runtime is polynomial in $|L|$ for fixed $n \ge 2$.

For List-Feasibility, we speed up the dynamic programming algorithm from Section 3, obtaining an algorithm with runtime $\mathcal{O}\big((|L|/n^2+1)^{n^2/2}\cdot n!\cdot n^3\cdot\min\{|L|,n^2\}\cdot\log|L|\big)$ and an algorithm with runtime $\mathcal{O}\big((n/\sqrt{8})^{n^2}\cdot n!\cdot n^5\log n+n^2\log|L|\big)$; see Section 4. The above runtimes are expressed in terms of the logarithmic cost model of computation to show explicitly how the runtimes depend on the length of L (actually on its largest entry).

Although we cannot characterize minimal feasible lists, we can bound their entries. Namely, we show that, in a minimal feasible list of order n, each swap occurs at most $n^2/4+1$ times; see Proposition 2. As a corollary, this yields that LIST-FEASIBILITY is in NP and fixed-parameter tractable with respect to n.

2 Complexity

Yamanaka et al. [21] showed that LIST-FEASIBILITY is NP-hard. In their reduction, however, some swaps have multiplicity $\Theta(n)$. In this section, we show that LIST-FEASIBILITY is NP-hard even if all swaps have multiplicity at most 8. In other words, we show that it is NP-hard to decide whether a list whose entries are all at most 8 is feasible. We reduce from POSITIVE NAE 3-SAT DIFF, a variant of NOT-ALL-EQUAL 3-SAT. Recall that in NOT-ALL-EQUAL 3-SAT one is given a Boolean formula in conjunctive normal form with three literals per clause and the task is to decide whether there exists a variable assignment such that in no clause all three literals have the same truth value. By Schaefer's dichotomy theorem [16], NOT-ALL-EQUAL 3-SAT is NP-hard even if no negative (i.e., negated) literals are admitted. In POSITIVE NAE 3-SAT DIFF, additionally each clause contains three different variables. We show that this variant is NP-hard, too.

Lemma 1 POSITIVE NAE 3-SAT DIFF *is NP-hard.*

Proof: We show the NP-hardness of POSITIVE NAE 3-SAT DIFF by reduction from NOT-ALL-EQUAL 3-SAT. Let Φ be an instance of NOT-ALL-EQUAL 3-SAT with variables $v_1, v_2, \dots, v_N$. First we show how to get rid of negative variables and then how to get rid of multiple occurrences of the same variable in a clause.

We create an instance Φ' of POSITIVE NAE 3-SAT DIFF as follows. For every variable v_i, we introduce two new variables x_i and y_i. We replace each occurrence of v_i by x_i and each occurrence of $\neg v_i$ by y_i. We need to force y_i to be $\neg x_i$. To this end, we introduce the clause $(x_i \vee y_i \vee y_i)$.

Now, we introduce three additional variables a, b, and c that form the clause $(a \vee b \vee c)$. Let $d = (x \vee x \vee y)$ be a clause that contains two occurrences of the same variable. Note that a clause that contains three occurences of the same variable is trivially not satisfiable. We replace d by three clauses $(x \vee y \vee a)$, $(x \vee y \vee b)$, $(x \vee y \vee c)$. Since at least one of the variables a, b, or c has to be true and at least one has to be false, x and y cannot have the same assignment, i.e., $x = \neg y$. Hence, Φ' is satisfiable if and only if Φ is. Clearly, the size of Φ' is polynomial in the size of Φ. □

Theorem 1 LIST-FEASIBILITY *is NP-complete; it is NP-hard even when restricted to lists whose entries are at most 8.*

Proof: We first show that LIST-FEASIBILITY is in NP, then that it is NP-hard.

Membership in NP. We proceed as indicated in the introduction. Given a list $L = (l_{ij})$, we guess a list $L' = (l'_{ij})$ with $2(L) = 2(L')$ and $l'_{ij} \le \min\{l_{ij}, n^2/4+1\}$ together with a permutation of its $\mathcal{O}(n^4)$ swaps. Then we can efficiently test whether we can apply the swaps in this order to id_n. If yes, then the list L' is feasible (and, due to $L' \to L$, a witness for the feasibility of L), otherwise we discard it. Note that L is feasible if and only if such a list L' exists and is feasible. One direction is obvious by the definition of minimal feasible lists. To show the other direction, we assume that L is feasible. Thus there exists a minimal feasible list $L' = (l_{ij})$ of type $2(L)$. By Proposition 2 (see Section 4), we have $l'_{ij} \le n^2/4 + 1$.

NP-hardness. We split the hardness proof, which uses gadgets for variables and clauses, into several parts. First, we introduce some notation, then we give the intuition behind our reduction. Next, we present our variable and clause gadgets in more detail. Finally, we show the correctness of the reduction.

Notation. Recall that we label the wires of a tangle by their index in the start permutation id_n. In particular, for a wire ε, its neighbor to the right is the wire $\varepsilon + 1$. If a wire γ is to the left of some other wire δ in the start permutation, then we write $\gamma < \delta$. If all wires in a set Γ are to the left of all wires in a set Δ in the start permutation, then we write $\Gamma < \Delta$. We use $\gamma < \Gamma$ as shorthand for $\{\gamma\} < \Gamma$ and $\Delta < \delta$ for $\Delta < \{\delta\}$. Correspondingly, if we say that wire γ swaps with the set Γ, it means that γ swaps with every wire in Γ (and hence γ and Γ change their order).

Setup. Given an instance $\Phi = d_1 \wedge \cdots \wedge d_M$ of Positive NAE 3-SAT Diff with clauses $d_1, \ldots, d_M$ and variables $w_1, \ldots, w_N$, we construct in polynomial time a list L of swaps such that there is a tangle T realizing L if and only if Φ is a yes-instance.

In L, we have two inner wires λ and $\lambda' = \lambda + 1$ that swap eight times. As in the proof of Section 1, we call the regions bounded by λ and λ' between two consecutive (λ, λ') swaps loops. Here, we also call the two regions bounded by λ and λ' and the horizontal lines that start or end the tangle loops. (see Fig. 4). We distinguish between two types of loops: four λ'–λ loops, where λ' is on the left and λ is on the right side, and five λ–λ' loops with λ on the left and λ' on the right side. In the following, we construct variable and clause gadgets. Each variable gadget will contain a specific wire that represents the variable, and each clause gadget will contain a specific wire that represents the clause. The corresponding variable and clause wires swap in one of the four λ'–λ loops. We call the first two λ'–λ loops *true-loops*, and the last two λ'–λ loops *false-loops*. If the corresponding variable is true, then the variable wire swaps with the corresponding clause wires in a true-loop, otherwise in a false-loop.

Apart from λ and λ', our list L contains (many) other wires, which we split into groups. For every $i \in [N]$, we introduce sets V_i and V'_i of wires that together form the gadget for variable w_i of Φ. These sets are ordered (initially) $V_N < V_{N-1} < \cdots < V_1 < \lambda < \lambda' < V'_1 < V'_2 < \cdots < V'_{N-1} < V'_N$; the order of the wires inside these sets will be detailed in the next two paragraphs. Let $V = V_1 \cup V_2 \cup \cdots \cup V_N$ and $V' = V'_1 \cup V'_2 \cup \cdots \cup V'_N$. Similarly, for every $j \in [M]$, we introduce a set C_j of wires that contains a *clause wire* c_j and three sets of wires D^1_j, D^2_j, and D^3_j that represent occurrences of variables in a clause d_j of Φ. The wires in C_j are ordered $D^3_j < D^2_j < D^1_j < c_j$. Together, the wires in $C = C_1 \cup C_2 \cup \cdots \cup C_M$ represent the clause gadgets; they are ordered $V < C_M < C_{M-1} < \cdots < C_1 < \lambda$. Additionally, our list L contains a set $E = \{\varphi_1, \ldots, \varphi_7\}$ of wires that will make our construction rigid enough. The order of all wires in L is $V < C < \lambda < \lambda' < E < V'$. Now we present our gadgets in more detail.

Variable gadget. First we describe the variable gadget, which is illustrated in Fig. 4. For every $i \in [N]$ (that is, for each variable w_i of Φ), we introduce two sets of wires, V_i and V'_i. Each set V'_i contains a *variable wire* v_i that has four swaps with λ and no swaps with λ'. Therefore, v_i intersects at least one and at most two λ'–λ loops. In order to prevent v_i from intersecting both a true- and a false-loop, we introduce two wires $\alpha_i \in V_i$ and $\alpha'_i \in V'_i$ with $\alpha_i < \lambda < \lambda' < \alpha'_i < v_i$; see Fig. 4. These wires neither swap with v_i nor with each other, but they have two swaps with both λ and λ'. We want to force α_i and α'_i to have the two true-loops on their right and the two

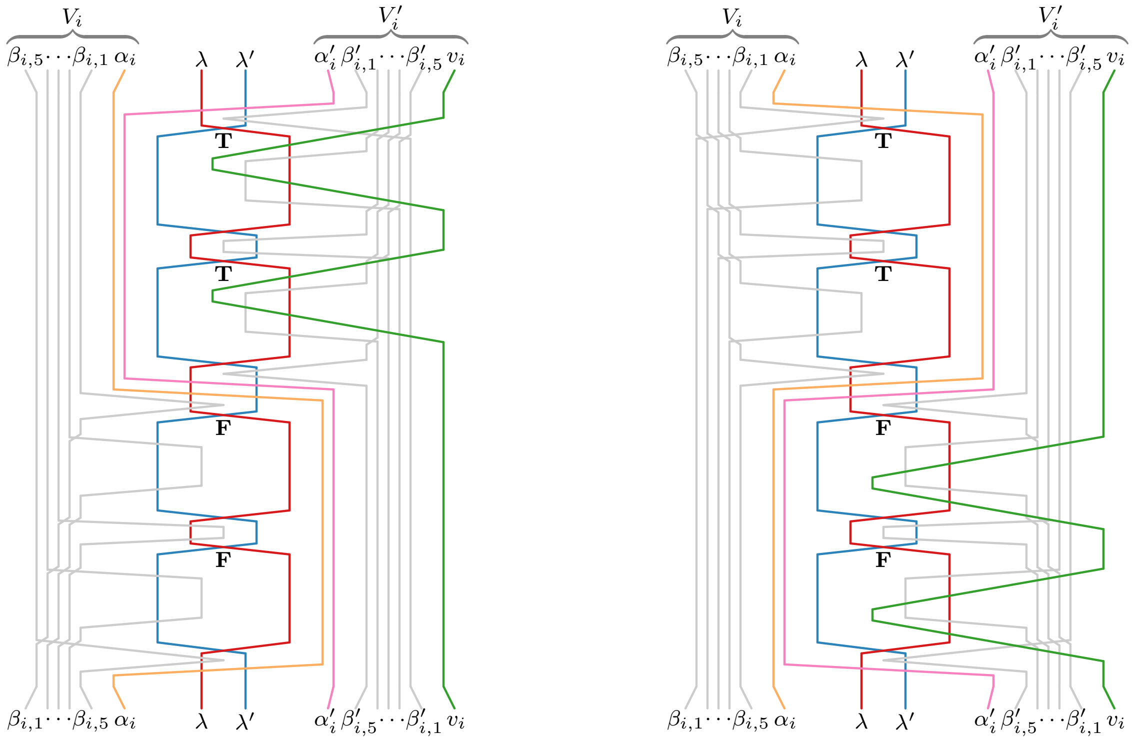

V_i
V'_i
$\beta_{i,5} \cdots \beta_{i,1}\ \alpha_i$
λ
λ'
$\alpha'_i\ \beta'_{i,1} \cdots \beta'_{i,5}\ v_i$
T
T
F
F
$\beta_{i,1} \cdots \beta_{i,5}\ \alpha_i$
$\alpha'_i\ \beta'_{i,5} \cdots \beta'_{i,1}\ v_i$

Figure 4: A variable gadget with a variable wire v_i that corresponds to the variable w_i that is true (left) or false (right). The λ'–λ loops are labeled **T** for true and **F** for false.

false-loops on their left, or vice versa. This will ensure that v_i cannot reach both a true- and a false-loop simultaneously.

To this end, we introduce, for $j \in [5]$, a *β_i-wire* $\beta_{i,j} \in V_i$ and a *β'_i-wire* $\beta'_{i,j} \in V'_i$. These are ordered $\beta_{i,5} < \beta_{i,4} < \cdots < \beta_{i,1} < \alpha_i$ and $\alpha'_i < \beta'_{i,1} < \beta'_{i,2} < \cdots < \beta'_{i,5} < v_i$. Every pair of β_i-wires as well as every pair of β'_i-wires swaps exactly once. Neither β_i- nor β'_i-wires swap with α_i or α'_i. Each β'_i-wire has four swaps with v_i. Moreover, $\beta_{i,1}, \beta_{i,3}, \beta_{i,5}, \beta'_{i,2}, \beta'_{i,4}$ swap with λ twice. Symmetrically, $\beta_{i,2}, \beta_{i,4}, \beta'_{i,1}, \beta'_{i,3}, \beta'_{i,5}$ swap with λ' twice; see Fig. 4.

We use the β_i- and β'_i-wires to fix the minimum number of λ'–λ loops that are on the left of α_i and on the right of α'_i, respectively. Note that, together with λ and λ', the β_i- and β'_i-wires have the same structure as the wires shown in Fig. 2. Due to Section 1, this structure is rigid in the sense that the order of the swaps along each wire is fixed.

Therefore, there is a unique order of swaps between the β_i-wires and λ or λ', i.e., for $j \in [4]$, every pair of $(\beta_{i,j+1}, \lambda)$ swaps (or $(\beta_{i,j+1}, \lambda')$ swaps, depending on the parity of j) can be done only after the pair of $(\beta_{i,j}, \lambda')$ swaps (or $(\beta_{i,j}, \lambda)$ swaps, respectively). We have the same rigid structure on the right side with β'_i-wires. Hence, there are at least two λ'–λ loops and three λ–λ' loops to the left of α_i and at least two λ'–λ loops and three λ–λ' loops to the right of α'_i. Since α_i and α'_i do not swap, there cannot be a λ'–λ loop that appears simultaneously on both sides. (Note that the third λ–λ' loop does appear on both sides.)

Note that the (λ, λ') swaps that belong to the same side need to be consecutive, otherwise α_i

or α'_i would have to swap more than twice with λ and λ'. Thus, there are only two ways to order the swaps among the wires α_i, α'_i, λ, λ'; the order is either

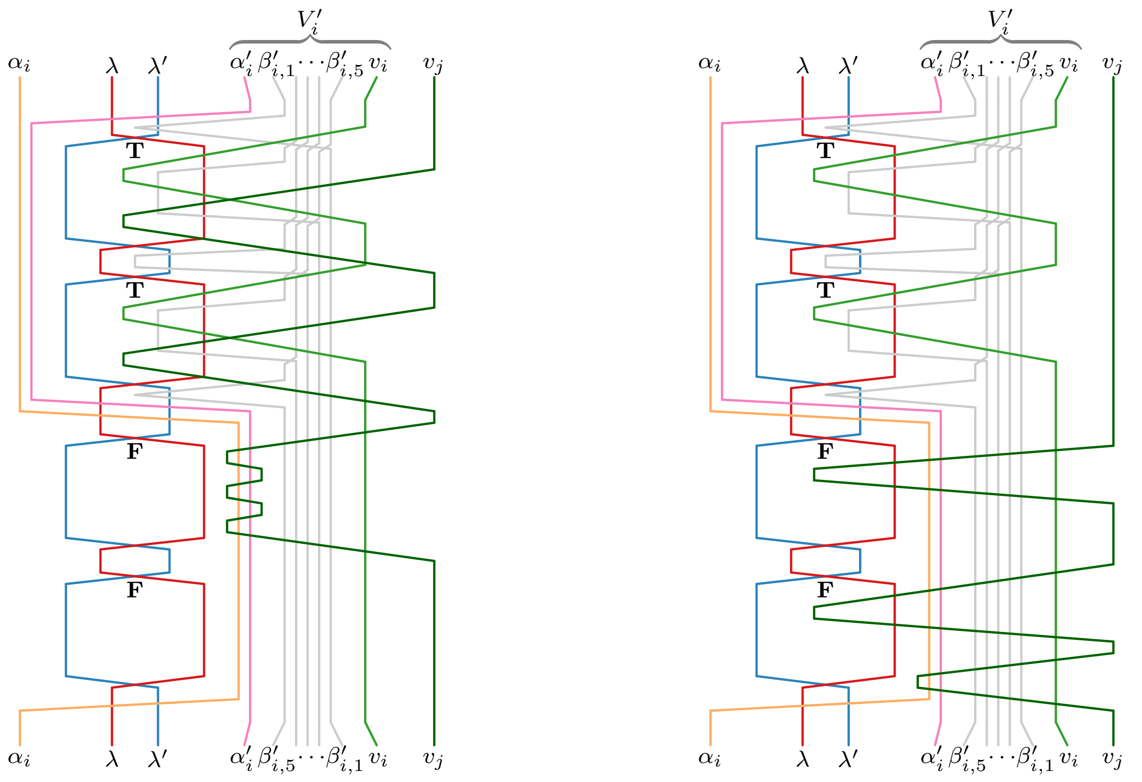
- (α'_i, λ'),
- (α'_i, λ),
- four times (λ, λ'),
- (α'_i, λ),
- (α'_i, λ') and (α_i, λ) in whichever order,
- (α_i, λ'),
- four times (λ, λ'),
- (α_i, λ'), and
- (α_i, λ)

(see Fig. 4 (left)) – or the reverse (see Fig. 4 (right)). It is easy to see that in the first case v_i can reach only the first two λ'–λ loops (the true-loops), and in the second case only the last two (the false-loops).

To avoid that the gadget for variable w_i restricts the proper functioning of the gadget for some variable w_j with $j > i$, we add the following swaps to L: for any $j > i$, α_j and α'_j swap with both V_i and V'_i twice, the β_j-wires swap with α'_i and V_i four times, and, symmetrically, the β'_j-wires swap with α_i and V'_i four times, and v_j swaps with α_i and V'_i six times.

We briefly explain these multiplicities. Wire α_j (α'_j, resp.) swaps with all wires in V_i and V'_i twice so that it reaches the corresponding λ–λ' or λ'–λ loops by first crossing all wires of V_i (V'_i). Then α_j (α'_j) crosses all wires of V'_i (V_i) and finally goes back crossing all of them a second time. This way, there are no restrictions for α_j (α'_j) regarding which λ'–λ loops it encloses. For the variable wire v_j, see Fig. 5. It swaps six times with α_i and V'_i, which guarantees that v_j can reach both upper or both lower λ'–λ loops. If w_i and w_j represent the same truth value, v_j needs a total of four swaps with each of the β'_i-wires to enter a loop and to go back. In this case, v_j can make the six swaps with α_i and α'_i next to the λ'–λ loops representing the other truth value. Then, v_j uses the two extra swaps with each of the β'_i-wires to reach α_i and α'_i; see Fig. 5 (left). If w_i and w_j represent distinct truth values, v_j needs a total of four swaps with each of the β'_i-wires and with α'_i and α_i to enter a loop and to go back. We can accommodate the two extra swaps with each of these wires afterwards; see Fig. 5 (right). For the β_j-wires (and β'_j-wires), we use the same argument as for v_j above, but each of these wires has only four swaps with each of the other wires in $V_i \cup \{\alpha'_i\}$ (or $V'_i \cup \{\alpha_i\}$) because a β-wire (or a β'-wire) needs to reach only one λ–λ' or λ'–λ loop instead of two (as v_j).

Clause gadget. For every clause d_j from Φ, $j \in [M]$, we introduce a set of wires C_j. It contains the clause wire c_j that has eight swaps with λ'. We want to force each c_j to appear in all λ'–λ loops. To this end, we use (once for all clause gadgets) the set E with the seven *φ-wires* $\varphi_1, \dots, \varphi_7$ ordered $\varphi_1 < \dots < \varphi_7$. They create a rigid structure according to Observation 1 similar to the β_i-wires. Each pair of φ-wires swaps exactly once. For each $k \in [7]$, if k is odd, then φ_k swaps twice with λ and twice with c_j for every $j \in [M]$. If k is even, then φ_k swaps twice with λ'. Since c_j does not swap with λ, each pair of swaps between c_j and a φ-wire with odd index appears inside a λ'–λ loop. Due to the rigid structure, each of these pairs of swaps occurs in a different λ'–λ loop; see Fig. 6.



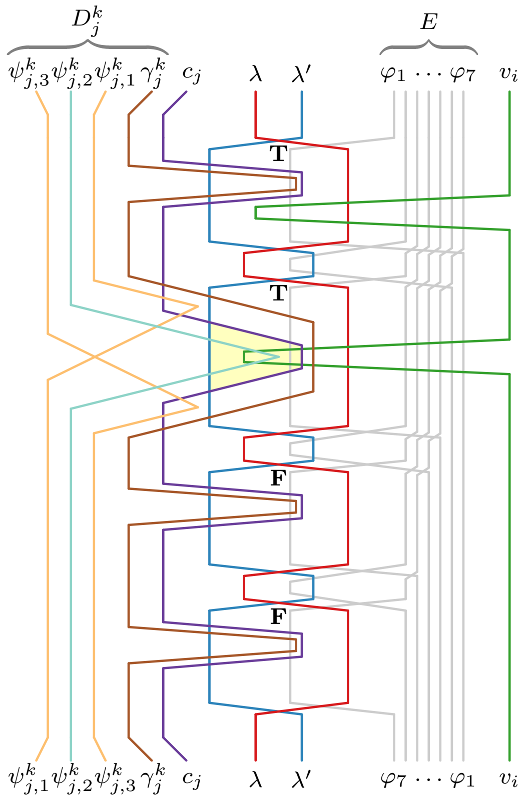
V'_i
α_i
λ
λ'
$\alpha'_i\,\beta'_{i,1}\cdots\beta'_{i,5}\,v_i$
v_j
T
T
F
F
α_i
λ
λ'
$\alpha'_i\,\beta'_{i,5}\cdots\beta'_{i,1}\,v_i$
v_j

Figure 5: A realization of swaps between the variable wire v_j and all wires that belong to the variable gadget corresponding to the variable w_i. On the left the variables w_i and w_j are both true, and on the right w_i is true, whereas w_j is false.

If a variable w_i belongs to a clause d_j, then our list L contains two (v_i, c_j) swaps. Since every clause has exactly three different positive variables, we want to force variable wires that belong to the same clause to swap with the corresponding clause wire in different λ'–λ loops. This way, every clause contains at least one true and at least one false variable if Φ is satisfiable.

We call the part of a clause wire c_j that is inside a λ'–λ loop an *arm* of c_j. We want to "*protect*" the arm that is intersected by a variable wire from other variable wires. To this end, for every occurrence $k \in [3]$ of a variable in d_j, we introduce four more wires. The wire γ_j^k will protect the arm of c_j that the variable wire of the k-th variable of d_j intersects. Below we detail how to realize this protection. For now, just note that, in order not to restrict the choice of the λ'–λ loop, γ_j^k swaps twice with φ_ℓ for every odd $\ell \in [7]$. Similarly to c_j, the wire γ_j^k has eight swaps with λ' and appears once in every λ'–λ loop. Additionally, γ_j^k has two swaps with c_j. In the first (and last) permutation, we have that $\gamma_j^k < c_j$.

We force γ_j^k to protect the correct arm in the following way. Consider the λ'–λ loop where an arm of c_j swaps with a variable wire v_i. We want the order of swaps along λ' inside this loop to be fixed as follows: λ' first swaps with γ_j^k, then twice with c_j, and then again with γ_j^k. This would prevent all variable wires that do not swap with γ_j^k from reaching the arm of c_j. To achieve this, we introduce three *ψ_j^k-wires* $\psi_{j,1}^k, \psi_{j,2}^k, \psi_{j,3}^k$ with the initial order $\psi_{j,3}^k < \psi_{j,2}^k < \psi_{j,1}^k < \gamma_j^k$.

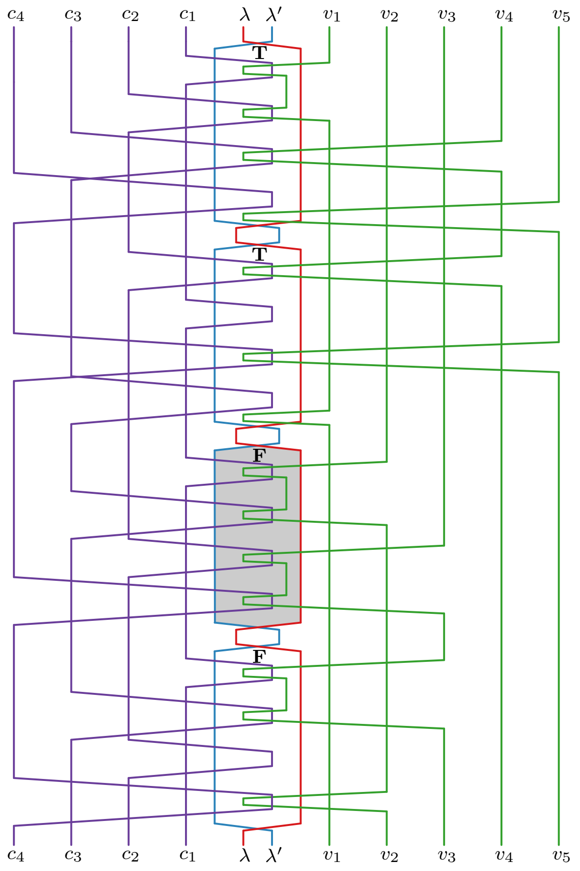


D_j^k
E
$\psi_{j,3}^k\,\psi_{j,2}^k\,\psi_{j,1}^k\,\gamma_j^k$
c_j
λ
λ'
$\varphi_1 \cdots \varphi_7$
v_i
T
T
F
F
$\psi_{j,1}^k\,\psi_{j,2}^k\,\psi_{j,3}^k\,\gamma_j^k$
c_j
λ
λ'
$\varphi_7 \cdots \varphi_1$
v_i

Figure 6: A gadget for clause c_j showing only one of the three variable wires, namely v_i. The region shaded in yellow is the arm of c_j that is protected from other variables by γ_j^k.

Each pair of ψ_j^k-wires swaps exactly once, $\psi_{j,1}^k$ and $\psi_{j,3}^k$ have two swaps with c_j, and $\psi_{j,2}^k$ has two swaps with λ' and v_i. Together with c_j and λ', the ψ_j^k-wires have the structure assumed in Observation 1 (similar to the φ- and β_i-wires), so the order of the above-mentioned swaps along each ψ_j^k-wire is unique. No ψ_j^k-wire swaps with γ_j^k. Also, since $\psi_{j,2}^k$ does not swap with c_j, the $(\psi_{j,2}^k, v_i)$ swaps can appear only inside the λ'–c_j loop that contains the arm of c_j we want to protect from other variable wires. Since c_j has to swap with $\psi_{j,1}^k$ before and with $\psi_{j,3}^k$ after the $(\psi_{j,2}^k, \lambda')$ swaps, and since there are only two swaps between γ_j^k and c_j, there is no way for any variable wire except for v_i to reach the arm of c_j without also intersecting γ_j^k; see Fig. 6.

Finally, for every $j \in [M]$ and every $k \in [3]$, let $D_j^k = \{\psi_{j,3}^k, \psi_{j,2}^k, \psi_{j,1}^k, \gamma_j^k\}$. The wires in D_j^k are initially in this order. We now consider the behavior of the wires among the sets D_j^1, D_j^2, and D_j^3, as well as the behavior of the wires in C_j with respect to other clause and variable gadgets. For every $k, \ell \in [3]$ with $k \neq \ell$, the ψ_j^k-wire has two swaps with the ψ_j^ℓ-wire in order to not restrict the ψ-wires to a specific λ'–λ loop. Furthermore, γ_j^k has four swaps with every ψ_j^ℓ-wire and two swaps with every wire γ_j^ℓ. Note that, if $k > \ell$, it suffices for γ_j^k to cross each γ_j^ℓ at the λ'–λ loop where γ_j^k "protects" c_j and to go back directly afterwards. Also, γ_j^k uses one swap to cross the ψ_j^ℓ-wires before the first λ'–λ loop, then, using two swaps, the ψ_j^ℓ-wires cross γ_j^k next to the λ'–λ loop where



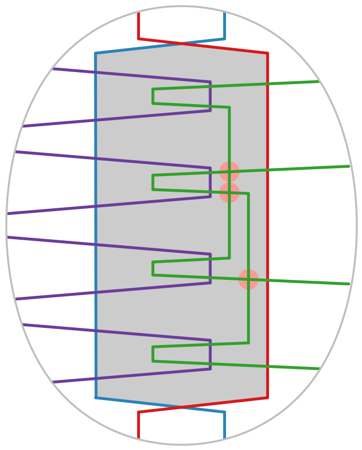


Figure 7: Tangle obtained from the satisfiable formula $\Phi = (w_1 \vee w_2 \vee w_3) \wedge (w_1 \vee w_3 \vee w_4) \wedge (w_2 \vee w_3 \vee w_4) \wedge (w_2 \vee w_3 \vee w_5)$. Here, w_1, w_4 and w_5 are set to true, whereas w_2 and w_3 are set to false. We show only λ, λ', and all variable and clause wires.
Inset: problems that occur if variable wires swap with clause wires in a different order.

γ_j^ℓ "protects" c_j, and γ_j^k uses the fourth swap to go back after the last λ'–λ loop.

For every $i < j$ and every $k \in [3]$, the wires c_j and γ_j^k have eight swaps with every wire in C_i, which allows c_j and γ_j^k to enter every λ'–λ loop and to go back. Similarly, every ψ_j^k-wire has two swaps with every wire in C_i, which allows the ψ_j^k-wire to reach one of the λ'–λ loops. Since all wires in V are to the left of all wires in C, each wire in C swaps twice with all wires in V (including the α-wires) and twice with all α'-wires. Finally, the φ-wires with odd index have (beside two swaps with every c_j) two swaps with every γ_j^k, and all φ-wires have, for every $i \in [N]$, two swaps with every wire in $(V_i' \setminus \{v_i\}) \cup \{\alpha_i\}$ and four swaps with every wire v_i to let the wires of the variable gadgets enter (or cross) the λ–λ' or λ'–λ loops.

Note that the order of the arms of the clause wires inside a λ'–λ loop cannot be chosen arbitrarily. If a variable wire intersects more than one clause wire, the arms of these clause wires occur consecutively, as for v_2 and v_3 in the shaded region in Fig. 7. If we had an interleaving pattern of variable wires (see inset), say v_2 first intersects c_1, then v_3 intersects c_2, then v_2 intersects c_3, and finally v_3 intersects c_4, then v_2 and v_3 would need more swaps than calculated in the analysis above.

Correctness. Clearly, if Φ is satisfiable, then there is a tangle obtained from Φ as described above that realizes the list L, so L is feasible; see Fig. 7 for an example. On the other hand, if there is a tangle that realizes the list L that we obtain from the reduction, then Φ is satisfiable. This follows from the rigid structure of a tangle that realizes L. The only flexibility is in which type of loop (true or false) a variable wire swaps with the corresponding clause wire. As described above, if a tangle exists, then each clause wire swaps with the corresponding three variable wires in three different loops (at least one of which is a true-loop and at least one is a false-loop). In this case, the position of the variable wires yields a truth assignment satisfying Φ. □

Note that our proof that shows that List-Feasibility is in NP does not show that the decision version of Tangle-Height Minimization is also in NP because the minimum height can be exponential in the size of the input.

3 Algorithms for Minimizing Tangle Height

The two algorithms that we describe in this section test whether a given list is feasible and, if yes, construct a height-optimal tangle realizing the list. We start with an observation and some definitions, then we present an algorithm for simple lists (see Section 3.1), and finally we describe an algorithm for general lists (see Section 3.2).

Let $(F_n)_{n\geq 1}$ be the Fibonacci sequence with $F_1 = F_2 = 1$ and, for $n \geq 3$, $F_n = F_{n-1} + F_{n-2}$.

Lemma 2 *Given a positive integer n and a permutation $\pi \in S_n$, the number of permutations adjacent to π is $F_{n+1} - 1$.*

Proof: By induction on n, we prove the slightly different claim that the set $P(\pi)$ consisting of π and its adjacent permutations has size F_{n+1}.

For $n = 1$, there is only one permutation '1' and, hence, $|P(1)| = 1$. Furthermore, for $n = 2$, there are only two permutations '12' and '21', which are also adjacent. Thus, $|P(12)| = |P(21)| = 2$. These are the Fibonacci numbers F_2 and F_3.

Let $n > 2$ and $\pi \in S_n$. Note that we can partition the permutations in $P(\pi)$ into two groups. The first group $P_1(\pi)$ contains the permutations of $P(\pi)$ where the last wire is the same as in π, and the second group $P_2(\pi)$ contains the permutations of $P(\pi)$ where the last wire is different from π. Clearly, $|P(\pi)| = |P_1(\pi)| + |P_2(\pi)|$.

To obtain $P_1(\pi)$, we first remove the last wire of π. This yields a permutation $\pi' \in S_{n-1}$ (after possibly renaming the wires to avoid a gap in our naming scheme). By our inductive hypothesis, $|P(\pi')| = F_n$. Then, we append the last wire of π to every permutation in $P(\pi')$. This yields $P_1(\pi)$.

For $P_2(\pi)$, observe that the last two wires in every permutation in $P_2(\pi)$ are swapped compared to π. We remove these two wires from π. This yields a permutation $\pi'' \in S_{n-2}$. Again by our inductive hypothesis, $|P(\pi'')| = F_{n-1}$. Then, we append the last two wires of π in swapped order to every permutation in $P(\pi'')$. This yields $P_2(\pi)$.

Summing up, we get $|P(\pi)| = F_n + F_{n-1} = F_{n+1}$. □

For a permutation $\pi \in S_n$ and a list $L = (l_{ij})$, we define the map $\pi L\colon [n] \to \mathbb{Z}$ by

$$\pi L(i) = \pi(i) + |\{j\colon \pi(i) < \pi(j) \leq n \text{ and } l_{ij} \text{ odd}\}| - |\{j\colon 1 \leq \pi(j) < \pi(i) \text{ and } l_{ij} \text{ odd}\}|. \quad (1)$$

A list L is called *π-consistent* if $\pi L \in S_n$, or, more rigorously, if πL induces a permutation of $[n]$. In this case, for each wire $i \in [n]$, $\pi L(i)$ is the final position of i, that is, the position after all swaps in L have been applied to π. An id_n-consistent list is *consistent*. For example, the list $L = \{(1,2),(2,3),(1,3)\}$ is consistent (because $\mathrm{id}_n L = \langle 3,2,1\rangle$), whereas the list $L' = \{(1,3)\}$ is not (because $\mathrm{id}_n L' = \langle 2,2,2\rangle$).

If a list is not consistent, then it is clearly not feasible. However, not all consistent lists are feasible. For example, the list $\{(1,3),(1,3)\}$ is consistent but not feasible. In Section 4, we show that consistency *is* sufficient for the feasibility of simple (and, more generally, for odd) lists. Clearly, an even list is always consistent. For a list $L = (l_{ij})$, recall that $1(L) = (l_{ij} \bmod 2)$. Since $\mathrm{id}_n L = \mathrm{id}_n 1(L)$, the list L is consistent if and only if $1(L)$ is consistent. We can compute $1(L)$ and check its consistency in $\mathcal{O}(n + |1(L)|) \subseteq \mathcal{O}(n^2)$ time (see Proposition 3 for details).

In the following, we define properties of lists given by permutations or tangles. We show that these lists behave as expected. For any permutation $\pi \in S_n$, we define the simple list $L(\pi) = (l_{ij})$ such that for $1 \le i < j \le n$, $l_{ij} = 0$ if $\pi(i) < \pi(j)$, and $l_{ij} = 1$ otherwise.

Lemma 3 *For every positive integer n and every permutation $\pi \in S_n$, $L(\pi)$ is the unique simple list with $\mathrm{id}_n L(\pi) = \pi$.*

Proof: Let $L = (l_{ij}) = L(\pi)$. By the definition in Eq. (1), $\mathrm{id}_n L$ is a map from $[n]$ to $\mathbb{Z}$ such that

$$\begin{aligned}
\mathrm{id}_n L(i) &= i + |\{j\colon i < j \le n \text{ and } \pi(i) > \pi(j)\}| - |\{j\colon 1 \le j < i \text{ and } \pi(i) < \pi(j)\}| \\
&= i + |\{j\colon i < j \le n \text{ and } \pi(i) > \pi(j)\}| + |\{j\colon 1 \le j < i \text{ and } \pi(i) > \pi(j)\}| \\
&\quad - |\{j\colon 1 \le j < i \text{ and } \pi(i) > \pi(j)\}| - |\{j\colon 1 \le j < i \text{ and } \pi(i) < \pi(j)\}| \\
&= i + |\{j\colon 1 \le j \le n \text{ and } \pi(i) > \pi(j)\}| - |\{j\colon 1 \le j < i\}| \\
&= i + (\pi(i) - 1) - (i - 1) = \pi(i).
\end{aligned}$$

Hence $\mathrm{id}_n L = \pi$. In particular, L is consistent.

We show the uniqueness of L by contradiction. Assume that there is a simple list $L' = (l'_{ij}) \neq L$ such that $\mathrm{id}_n L' = \pi$ (and so L' is consistent). Hence there exists a pair $(i,j) \in [n]^2$ such that $l'_{ij} \neq l_{ij}$. Since both L and L' are simple, this means that $l'_{ij} = 1 - l_{ij}$. Since L is consistent, $\mathrm{id}_n L(i)$ and $\mathrm{id}_n L(j)$ are the final positions of wires i and j, respectively. Since L' is also consistent, an analogous statement holds for L'. Given that $l'_{ij} = 1 - l_{ij}$, wires i and j are ordered differently in $\mathrm{id}_n L$ and $\mathrm{id}_n L'$. This contradicts our assumption that $\mathrm{id}_n L = \pi = \mathrm{id}_n L'$. Therefore, L is the unique simple list with $\mathrm{id}_n L = \pi$. □

Given a tangle T with n wires, we define the list $L(T) = (l_{ij})$, where, for every pair $(i,j) \in [n]^2$, l_{ij} is the number of occurrences of swap (i,j) in T.

Lemma 4 *For every tangle $T = \langle \pi_1, \pi_2, \ldots, \pi_h\rangle$ with $\pi_1 = \mathrm{id}_n$, we have $\pi_1 L(T) = \pi_h$.*

Proof: For every pair $(i,j) \in [n]^2$ with $i < j$, we have that l_{ij} is odd if and only if $\pi_h(i) > \pi_h(j)$. Hence $1(L(T)) = L(\pi_h)$. Now we obtain $\pi_1 L(T) = \mathrm{id}_n L(T) = \mathrm{id}_n 1(L(T)) = \mathrm{id}_n L(\pi_h) = \pi_h$, using the definition in Eq. (1) for the second equality and Section 3 for the last equality. □

3.1 Simple Lists

We first solve TANGLE-HEIGHT MINIMIZATION for simple lists. Our algorithm does BFS in an appropriately defined auxiliary graph.

Theorem 2 *For a simple list of order n,* Tangle-Height Minimization *can be solved in $\mathcal{O}(n!\varphi^n)$ time, where $\varphi = (\sqrt{5}+1)/2 \approx 1.618$ is the golden ratio.*

Proof: Let L be a consistent simple list. Wang's algorithm [19] creates a *simple* tangle from $\mathrm{id}_n\, L$, i.e., a tangle where all its permutations are distinct. Thus L is feasible. Note that the height of a simple tangle is at most $n!$. Let $T = (\mathrm{id}_n {=} \pi_1, \pi_2, \dots, \pi_h {=} \mathrm{id}_n\, L)$ be any tangle such that $L(T)$ is simple. Then, by Section 3, $\mathrm{id}_n\, L(T) = \pi_h$. By Section 3, $L(\pi_h)$ is the unique simple list with $\mathrm{id}_n\, L(\pi_h) = \pi_h = \mathrm{id}_n\, L$, so $L(T) = L(\pi_h) = L$ and thus T is a realization of L.

We compute an optimal tangle realizing $L = (l_{ij})$ as follows. Consider the graph G_L whose vertex set $V(G_L)$ consists of all permutations $\pi \in S_n$ with $L(\pi) \le L$ (componentwise). A directed edge (π, σ) between vertices π and σ in $V(G_L)$ exists if and only if π and σ are adjacent permutations and $L(\pi) \cap L(\pi^{-1}\sigma) = \varnothing$; the latter means that the set of (disjoint) swaps that transforms π to σ cannot contain swaps from the set that transforms id_n to π (otherwise the list that transforms id_n via π to σ would not be simple). The graph G_L has at most $n!$ vertices and maximum degree $F_{n+1} - 1$; see Section 3. It is well known that, for every positive integer n, $F_n = (\varphi^n - (-\varphi)^{-n})/\sqrt{5}$. Hence $F_n \in \Theta(\varphi^n)$. Furthermore, for each $h \ge 0$, there is a natural bijection between tangles of height $h+1$ realizing L and paths of length h in the graph G_L from the start permutation id_n to the permutation $\mathrm{id}_n\, L$. A shortest such path can be found by BFS in $\mathcal{O}(E(G_L)) = \mathcal{O}(n!\varphi^n)$ time. □

3.2 General Lists

Now we solve Tangle-Height Minimization for general lists. We use a dynamic program (DP).

Theorem 3 *For a list L of order n, the problem* Tangle-Height Minimization *can be solved in $\mathcal{O}((|L|/n^2+1)^{n^2/2} \cdot \varphi^n \cdot n! \cdot n \cdot \min\{|L|, n^2\} \cdot \log|L|)$ time, where $\varphi = (\sqrt{5}+1)/2 \approx 1.618$ is the golden ratio.*

Proof: Let $L = (l_{ij})$ be the given list. We describe a DP that computes the height of an optimal tangle realizing L (if it exists). It is not difficult to adjust the DP to also compute an optimal tangle. Let λ be the number of distinct consistent sublists of L. A sublist L' of L is, by definition, consistent if $\mathrm{id}_n\, L' \in S_n$.

Let $\pi \in S_n$ be any permutation, and let $L' = (l'_{ij})$ be any sublist of L such that $\mathrm{id}_n\, L' = \pi$. Then $\mathrm{id}_n\, 1(L') = \mathrm{id}_n\, L' = \pi$. Since the list $1(L')$ is simple, by Section 3, $1(L') = L(\pi)$. So for each $(i,j) \in [n]^2$ with $i < j$, the parity of (l'_{ij}) is determined by π. Hence, there are at most $\lfloor l_{ij}/2 \rfloor + 1$ choices for l'_{ij}. Therefore, the number of sublists L' of L with $\mathrm{id}_n\, L' = \pi$ is at most $P = \prod_{i<j}(\lfloor l_{ij}/2 \rfloor + 1)$ and the number λ of distinct consistent sublists of L is at most $|S_n| P = n! P$.

The DP computes a table H of size λ where, for any consistent sublist L' of L, $H(L')$ is the optimal height of a tangle realizing L' if L' is feasible (otherwise $H(L') = \infty$). We compute the entries of H in non-decreasing order of list length. All λ consistent sublists of L can be generated based on each permutation of S_n in overall $\mathcal{O}(\lambda n^2 \log|L|)$ time, and can be sorted in the same time as the list lengths are bounded by $|L| \le \lambda$. The second factor and the third factor in the running time come from the fact that each list consists of $\mathcal{O}(n^2)$ numbers between 0 and $|L|$. Note that the term does not appear in the overall running time, as it is only required as a preprocessing step, and the running time of the DP dominates the running time for generating these lists.

Let L' be the next sublist of L to consider. We compute the optimal height $H(L')$ of L'. To this end, let $\rho = \mathrm{id}_n\, 1(L') = \mathrm{id}_n\, L'$. This is the final permutation of any tangle that realizes L'. Now we go through the set S_ρ of permutations that are adjacent to ρ. According to Section 3,

$|S_\rho| = F_{n+1} - 1$. For each permutation $\pi \in S_\rho$, the set $L(\langle \pi, \rho \rangle)$ is the set of disjoint swaps that transforms π into ρ. Then

$$H(L') = \min_{\pi \in S_\rho} H(L' - L(\langle \pi, \rho \rangle)) + 1. \tag{2}$$

After we have filled the whole table H, the entry $H(L)$ contains the desired height of a realization of L (or ∞ if L is not feasible).

Computing the minimum in Equation (2) needs $F_{n+1} - 1$ lookups in the table H. We can iterate through the set S_ρ of adjacent permutations in $\mathcal{O}(n)$ time per permutation as outlined in the inductive procedure in the proof of Section 3. We can store H as an $\binom{n}{2}$-dimensional array—where each dimension corresponds to the number of swaps between a pair of wires—to look up the solution for a single permutation $\pi \in S_\rho$ in $\mathcal{O}(n^2 \log |L|)$ time. The table uses $\mathcal{O}(\lambda \log |L|)$ space: it has λ possible entries, and for each entry we need $\log |L|$ space to store it (and thus also $\log |L|$ time to read it). Since we have to consider only pairs of indices that have at least one swap in L, we can reduce the number of dimensions to $|L|$ if $|L| < \binom{n}{2}$. Thus, we can compute the table entry for each sublist in $\mathcal{O}\big(n^2 \log |L| + (F_{n+1} - 1) \cdot n \cdot \min\{|L|, n^2\} \cdot \log |L|\big) = \mathcal{O}\big((F_{n+1} - 1) \cdot n \cdot \min\{|L|, n^2\} \cdot \log |L|\big)$ time, and we spend $\mathcal{O}\big(\lambda (F_{n+1} - 1) \cdot n \cdot \min\{|L|, n^2\} \cdot \log |L|\big)$ time in total.

Assuming that $n \geq 2$, we bound λ as follows, using the inequality of arithmetic and geometric means (AGM) and Bernoulli's inequality (B). Due to Bernoulli, $1 + ry \leq (1 + y)^r$ for every real $y \geq -1$ and for every real $r \geq 1$. We use his inequality for $y = |L|/n^2$ and $r = n/(n-1)$:

$$\lambda \leq n! \prod_{i<j} (l_{ij}/2 + 1) \overset{\text{AGM}}{\leq} n! \left(\frac{\sum_{i<j} (l_{ij}/2 + 1)}{\binom{n}{2}} \right)^{\binom{n}{2}} = n! \left(\frac{|L|}{n(n-1)} + 1 \right)^{\binom{n}{2}} \overset{\text{B}}{\leq} n! \left(\frac{|L|}{n^2} + 1 \right)^{n^2/2}.$$

□

Note that, since $x + 1 \leq e^x$ for any $x \in \mathbb{R}$ (here $x = |L|/n^2$), the running time $\mathcal{O}((|L|/n^2 + 1)^{n^2/2} \cdot \varphi^n \cdot n! \cdot n \cdot \min\{|L|, n^2\} \cdot \log |L|)$ of the DP is upper-bounded by $\mathcal{O}(e^{|L|/2} \cdot \varphi^n \cdot n! \cdot n \cdot \min\{|L|, n^2\} \cdot \log |L|)$.

4 Algorithms for Deciding Feasibility

We investigate the feasibility problem for different classes of lists in this section. First we consider general lists, then simple lists, odd lists and, finally, even lists. Recall that a list L is even if and only if $1(L)$ is the zero list and L is odd if and only if $1(L) = 2(L)$.

For any list to be feasible, each triple of wires $i < j < k$ requires an (i, j) or a (j, k) swap if there is an (i, k) swap – otherwise wires i and k would be separated by wire j in any tangle. We call a list fulfilling this property *non-separable*. For odd lists, non-separability is implied by consistency (because consistency is sufficient for feasibility, see Proposition 4 in the following). The NP-hardness reduction from Section 2 shows that a non-separable list can fail to be feasible even when it is consistent.

4.1 General Lists

Our DP for Tangle-Height Minimization runs in $\mathcal{O}\big((|L|/n^2 + 1)^{n^2/2} \cdot \varphi^n \cdot n! \cdot n \cdot \min\{|L|, n^2\} \cdot \log |L|\big)$ time. We adjust this algorithm to the task of testing feasibility, which makes the algorithm simpler and faster. Then we bound the entries of minimal feasible lists (defined in Section 1) and use this bound to turn our exact algorithm into a fixed-parameter algorithm where the parameter is the number of wires (i.e., n).

Theorem 4 *There is an algorithm that, given a list L of order n, tests whether L is feasible in $\mathcal{O}\big((|L|/n^2+1)^{n^2/2} \cdot n! \cdot n^3 \cdot \min\{|L|, n^2\} \cdot \log|L|\big)$ time.*

Proof: Let F be a Boolean table with one entry for each consistent sublist L' of L such that $F(L') =$ `true` if and only if L' is feasible. This table can be filled by means of a dynamic programming recursion. The empty list is feasible. Let L' be a consistent sublist of L with $|L'| \geq 1$ and assume that for each strict sublist of L', the corresponding entry in F has already been determined. A sublist $\tilde{L}$ of L is feasible if and only if there is a realizing tangle of $\tilde{L}$ of height $|\tilde{L}|+1$. For each (i,j) swap in L', we check if there is a tangle realizing L' of height $|L'|+1$ such that (i,j) is the last swap. If no such swap exists, then L' is infeasible, otherwise it is feasible. To perform the check for a particular (i,j) swap, we consider the strict sublist L'' of L' that is identical to L' except an (i,j) swap is missing. If L'' is not consistent, we continue with the next swap. Otherwise, we check the entry $F(L'')$. If $F(L'') =$ `true`, we compute the final positions of i and j with respect to L'' (see Section 3). The desired tangle exists if and only if these positions differ by exactly one. In this case, we set $F(L') =$ `true`. Otherwise, we continue with the next swap. If there is no such swap, we set $F(L') =$ `false`.

The number of consistent sublists of L is upperbounded by $(|L|/n^2+1)^{n^2/2} \cdot n!$, see the proof of Section 3.2. For each consistent sublist, we have to check $\mathcal{O}(n^2)$ swaps. To check a swap, we have to compute the final positions of two wires, which can be done in $\mathcal{O}(n \log|L|)$ time. We can store F as a $\min\{|L|, \binom{n}{2}\}$-dimensional array where each dimension represents a potential swap occurring in L. Hence, addressing a cell in L takes time $\mathcal{O}(\min\{|L|, n^2\})$, which yields an overall running time of $\mathcal{O}\big((|L|/n^2+1)^{n^2/2} \cdot n! \cdot n^3 \cdot \min\{|L|, n^2\} \log|L|\big)$. □

Next we introduce a tangle-shortening construction. We use the following lemma that follows from odd-even sort and is well known [12].

Lemma 5 *For each integer $n \geq 2$ and each pair of permutations $\pi, \sigma \in S_n$, we can construct in $\mathcal{O}(n^2)$ time a tangle T of height at most $n+1$ that starts with π, ends in σ, and whose list $L(T)$ is simple.*

We use Section 4.1 in order to shorten lists without changing their type.

Lemma 6 *Let n be a positive integer, let $T = \langle \pi_1, \pi_2 \dots, \pi_h \rangle$ be a tangle of n wires, let $L = L(T) = (l_{ij})$, and let $P \subseteq \{\pi_1, \pi_2, \dots, \pi_h\}$ such that $\{\pi_1, \pi_h\} \subseteq P$. If, for every $1 \leq i < j \leq n$ and $l_{ij} > 0$, there exists a permutation $\pi \in P$ with $\pi(j) < \pi(i)$, then we can construct a tangle T' with $L' = L(T') = (l'_{ij})$ such that $l'_{ij} \leq \min\{l_{ij}, |P|-1\}$ and $2(L') = 2(L)$.*

Proof: We construct the tangle T' as follows. Let $p = |P|$ and consider the elements of P in the order of first occurrence in T, that is, $P = \{\pi'_1, \pi'_2, \dots, \pi'_p\} = \{\pi_{m_1}, \pi_{m_2}, \dots, \pi_{m_p}\}$ such that $1 = m_1 < m_2 < \dots < m_p = h$. For every two consecutive elements π'_k and π'_{k+1} with $k \in [p-1]$, we create a tangle T'_k that starts from π'_k, ends at π'_{k+1}, and whose list $L(T'_k) = (l'_{k,ij})$ is simple. Note that, by Section 4.1, we can always construct such a tangle of height at most $n+1$, where n is the number of wires in T. Now let $T' = \langle T'_1, T'_2, \dots, T'_{p-1} \rangle$. For each $k \in [p-1]$, let T_k be the subtangle of T that starts at π'_k and ends at π'_{k+1}. Note that both these permutations are in T. Let $L(T_k) = (l_{k,ij})$. The simplicity of the list $L(T'_k)$ implies that, for every $i, j \in [n]$, $l'_{k,ij} \leq l_{k,ij}$. Then, for every $i, j \in [n]$, $l'_{ij} \leq \min\{l_{ij}, p-1\}$. Since the tangles T' and T have common start and final permutations, for every $i, j \in [n]$, the numbers l_{ij} and l'_{ij} have the same parity, that is, $1(L') = 1(L)$. Recall that, for every $1 \leq i < j \leq n$ and $l_{ij} > 0$, there exists a permutation $\pi \in P$ with $\pi(j) < \pi(i)$. In other words, if $l_{ij} > 0$, then $l'_{ij} > 0$. Hence, $2(L') = 2(L)$. □

We want to upperbound the entries of a minimal feasible list. We first give a simple bound, which we then improve by a factor of roughly 2.

Proposition 1 *If $L = (l_{ij})$ is a minimal feasible list of order n, then $l_{ij} \leq \binom{n}{2} + 1$ for each $i, j \in [n]$.*

Proof: The list L is feasible, so there is a tangle $T = \langle \pi_1, \pi_2, \dots, \pi_h \rangle$ realizing L. We construct a set P of permutations of T, starting with $P = \{\pi_1, \pi_h\}$. Then, for each pair (i, j) with $1 \leq i < j \leq n$ and $l_{ij} > 0$, we add to P a permutation π_{ij} from T such that $\pi_{ij}(j) = \pi_{ij}(i) + 1$. Note that T must contain such a permutation since we assume that $l_{ij} > 0$ and that $\pi_1 = \mathrm{id}_n$. Clearly, $|P| \leq \binom{n}{2} + 2$.

Now Section 4.1 yields a tangle T' with $L' = L(T') = (l'_{ij})$ such that $l'_{ij} \leq \min\{l_{ij}, |P| - 1\}$ and $2(L') = 2(L)$. Hence, $L' \to L$. The list L is minimal, so $L = L'$. Thus, since $|P| \leq \binom{n}{2} + 2$, $l_{ij} = l'_{ij} \leq |P| - 1 \leq \binom{n}{2} + 1$ for each $i, j \in [n]$. □

Now we improve this bound by a factor of roughly 2.

Proposition 2 *If $L = (l_{ij})$ is a minimal feasible list of order n, then $l_{ij} \leq n^2/4 + 1$ for each $i, j \in [n]$.*

Proof: The list L is feasible, so there is a tangle $T = \langle \pi_1, \pi_2, \dots, \pi_h \rangle$ realizing L. We again construct a set P of permutations of T, starting from $P = \{\pi_1, \pi_h\}$ as follows.

Let G be the graph with vertex set $[n]$ that has an edge for each pair $\{i, j\}$ with $l_{ij} > 0$. The edge ij, if it exists, has weight $|i - j|$. For an edge ij with $i < j$, we say that a permutation π of T *witnesses* the edge ij of G (or the swap (i, j) of L) if $\pi(j) < \pi(i)$.

We repeat the following step until every edge of G is colored. Pick any non-colored edge of maximum weight and color it red. Let i and j with $i < j$ be the endpoints of this edge. Since $l_{ij} > 0$ and the tangle T realizes the list L, T contains a permutation π with $\pi(j) < \pi(i)$. Note that π witnesses the edge ij. Add π to P. Now for each k with $i < k < j$, do the following. Since $\pi_1 = \mathrm{id}_n$ and $\pi(j) < \pi(i)$, it holds that $\pi(k) < \pi(i)$ or $\pi(j) < \pi(k)$ (or both). In other words, π witnesses the edge ki, the edge jk, or both. We color each witnessed edge blue.

The coloring algorithm ensures that the graph G has no red triangles, so, by Turán's theorem [18], G has at most $n^2/4$ red edges. Hence $|P| \leq 2 + n^2/4$. By construction, every edge of G is witnessed by a permutation in P. Thus, Section 4.1 can be applied to P. This yields a tangle T' with $L' = L(T') = (l'_{ij})$ such that $l'_{ij} \leq \min\{l_{ij}, |P| - 1\}$ and $2(L') = 2(L)$. Hence, $L' \to L$. The list L is minimal, therefore $L' = L$. Since each entry of the list L' is at most $|P| - 1 \leq n^2/4 + 1$, the same holds for the entries of the list L. □

Combining Proposition 2 and our exact algorithm from Section 4.1 yields a fixed-parameter tractable algorithm with respect to n.

Theorem 5 *There is a fixed-parameter algorithm for* LIST-FEASIBILITY *with respect to the parameter n. Given a list L of order n, the algorithm tests whether L is feasible in $\mathcal{O}\big((n/\sqrt{8})^{n^2} \cdot n! \cdot n^5 \log n + n^2 \log |L|\big)$ time.*

Proof: Given the list $L = (l_{ij})$, let $L' = (l'_{ij})$ with $l'_{ij} = \min\{l_{ij}, n^2/4 + 1\}$ for each $i, j \in [n]$. We use our exact algorithm described in the proof of Section 4.1 to check whether the list L' is feasible. Since our algorithm checks the feasibility of every consistent sublist L'' of L', it suffices to combine this with checking whether $2(L'') = 2(L)$. If we find a feasible sublist L'' of the same type as L, then, by Proposition 2, L is feasible; otherwise, L is infeasible. Checking the type of L'' is easy. The runtime for this check is dominated by the runtime for checking the feasibility of L''. Constructing the list L' takes $\mathcal{O}(n^2 \log |L|)$ time. Note that $|L'| \leq \binom{n}{2} \cdot (n^2/4+1) = n^4/8 + n^2/2 - n^3/8 - n/2$. For $n \geq 12$, $3n^2/2 - n^3/8 - n/2 \leq 0$; hence $|L'| \leq n^4/8 - n^2$. Plugging this bound on $|L'|$ for $|L|$ into the runtime $\mathcal{O}\big((|L|/n^2+1)^{n^2/2} \cdot n! \cdot n^3 \cdot \min\{|L|, n^2\} \log |L|\big)$ of our exact algorithm (Section 4.1) yields a total runtime of $\mathcal{O}\big((n/\sqrt{8})^{n^2} \cdot n! \cdot n^5 \log n\big)$, plus the time $\mathcal{O}(n^2 \log |L|)$ needed to construct the list L' from the given list L. □

4.2 Simple Lists

If we restrict our study to simple lists, we can easily decide feasibility.

Proposition 3 *A simple list L is feasible if and only if L is consistent. Thus, we can check the feasibility of L in $\mathcal{O}(n + |L|)$ time, where n is the order of L.*

Proof: Clearly, if L is feasible, then L is also consistent. If L is consistent, then $\mathrm{id}_n L$ is a permutation. By Section 4.1, there exists a tangle T which starts from id_n, ends at $\mathrm{id}_n L$, and the list $L(T)$ is simple. By Section 3, $\mathrm{id}_n L(T) = \mathrm{id}_n L$. By Section 3, $L(T) = L$. So L is also feasible. We can check the consistency of L in $\mathcal{O}(n + |L|)$ time, which is equivalent to checking the feasibility of L. □

Odd Lists

For odd lists, feasibility reduces to that of simple lists. For a list l and a set $A \subseteq [n]$ of wire labels, let L_A be the $(|A| \times |A|)$-submatrix of L that consists of the rows and columns indexed by A.

Proposition 4 *For $n \geq 3$ and an odd list L of order n, the following statements are equivalent:*

1. *The list L is feasible.*
2. *The list $1(L)$ is feasible.*
3. *For each triple $A \subseteq [n]$, the list L_A is feasible.*
4. *For each triple $A \subseteq [n]$, the list $1(L_A)$ is feasible.*
5. *The list L is consistent.*
6. *The list $1(L)$ is consistent.*
7. *For each triple $A \subseteq [n]$, the list L_A is consistent.*
8. *For each triple $A \subseteq [n]$, the list $1(L_A)$ is consistent.*

Proof: We prove the proposition by proving three cycles of implications $1 \Rightarrow 5 \Rightarrow 6 \Rightarrow 2 \Rightarrow 1$, $3 \Rightarrow 7 \Rightarrow 8 \Rightarrow 4 \Rightarrow 3$, and $1 \Rightarrow 3 \Rightarrow 2 \Rightarrow 1$.

$1 \Rightarrow 5$. Clearly, all feasible lists are consistent.

$5 \Rightarrow 6$. Consistency of L means that $\mathrm{id}_n L \in S_n$. Since $\mathrm{id}_n 1(L) = \mathrm{id}_n L$, the list $1(L)$ is consistent, too.

$6 \Rightarrow 2$. Follows from Proposition 3 because the list $1(L)$ is simple.

$2 \Rightarrow 1$. We decompose L into $1(L)$ and $L' = (L - 1(L))$. Note that $L' = (l'_{ij})$ is an even list. Let $(i, j) \in L'$. Then $(i, j) \in 1(L)$ because L is odd. Consider a tangle T realizing $1(L)$. Let π be the layer in T where the swap (i, j) occurs. Just below π, insert l'_{ij} new layers such that the difference between one such layer and its previous layer is only the swap (i, j). Observe that every second new layer equals π – in particular the last one, which means that we can continue the tangle with the remainder of T. Applying this operation to all swaps in L' yields a tangle realizing L.

$3 \Rightarrow 7$. Clearly, all feasible lists are consistent.

$7 \Rightarrow 8$. Follows from the equality $\mathrm{id}_n\, 1(L_A) = \mathrm{id}_n\, L_A$.

$8 \Rightarrow 4$. Follows from Proposition 3, because the list $1(L_A)$ is simple.

$4 \Rightarrow 3$. For every triple $A \subseteq [n]$, we can argue as in the proof ($2 \Rightarrow 1$).

$1 \Rightarrow 3$. Trivial.

$3 \Rightarrow 2$. Let $1 \le i < k < j \le n$. By the equivalence ($1 \Leftrightarrow 2$), the odd list $L_{\{i,k,j\}}$ is infeasible if and only if $1(L_{\{i,k,j\}})$ is infeasible, that is, either $(i, j) \in L$ and (i, k), $(k, j) \notin L$, or $(i, j) \notin L$ and (i, k), $(k, j) \in L$. Define a binary relation $\le'$ on the set $[n]$ by letting $i \le' j$ if and only if either $i \le j$ and $(i, j) \notin L$, or $i > j$ and $(i, j) \in L$. (Note that $\le'$ encodes the final permutation of a tangle that realizes L.) Using the feasibility of L_A for all triples $A \subseteq [n]$, it follows that $\le'$ is a linear order. Let π be the (unique) permutation of the set $[n]$ such that $\pi^{-1}(1) \le' \pi^{-1}(2) \le' \cdots \le' \pi^{-1}(n)$. Observe that $L(\pi) = 1(L)$, so the list $1(L)$ is feasible.

□

Note that, for a feasible list L, it does not necessarily hold that $2(L)$ is feasible; see, e.g., for $n \ge 5$, the list L_n from Observation 1.

Even Lists

An even list is always consistent since it does not contain an odd number of swaps and its final permutation is the same as its start permutation. We show that, for sufficiently "rich" lists, non-separability is sufficient for an even list to be feasible, but in general this is not true.

Proposition 5 *Let n be a positive integer, and let $L = (l_{ij})$ be a non-separable even list of order n. If, for every $(i, j) \in [n]^2$, it holds that $l_{ij} \ge n - 1$ or $l_{ij} = 0$, then L is feasible.*

Proof: On the set $[n]$ of wires, we define a binary relation $\le_L$ as follows. For each $i, j \in [n]$, we set $i \le_L j$ if $i \le j$ and $l_{ij} = 0$; otherwise, i and j are incomparable. Since the list L is non-separable, the relation $\le_L$ is transitive, so it is a partial order. The dimension d of a partial order $\le_L$ on the set $[n]$ is the smallest number d of *linear orders* $\le_1, \ldots, \le_d$ of $[n]$ such that, for each $i, j \in [n]$, we have $i \le_L j$ if and only if, for every $k \in [d]$, it holds that $i \le_k j$. In other words, $\le_L$ can be seen as the intersection of $\le_1, \ldots, \le_d$. It is known that the dimension d of a poset on $[n]$ is at most $\lceil n/2 \rceil$ [9]; we will use $2d \le n + 1$ below. For each linear order $\le_k$ with $k \in [d]$, let π_k be the (unique) permutation of the set $[n]$ such that $\pi_k^{-1}(1) \le_k \pi_k^{-1}(2) \le_k \cdots \le_k \pi_k^{-1}(n)$ and $L_k = L(\pi_k)$. As a consequence of Section 3, the list L_k is feasible. Observe that $\mathrm{id}_n(L_k + L_k) = \mathrm{id}_n$ and that $(L_k + L_k)$ is even, where $+$ is the usual matrix addition. Therefore, $L' = L_1 + L_1 + L_2 + L_2 + \cdots + L_d + L_d$ is also feasible and even. Let $L' = (l'_{ij})$ and let $1 \le i < j \le n$.

If $l'_{ij} = 0$ then, for every $k \in [d]$, it holds that $\pi_k(i) < \pi_k(j)$, hence, $i \leq_k j$, which means that $i \leq_L j$ and $l_{ij} = 0$.

Otherwise, i and j are incomparable in $\leq_L$ and, hence, in at least one linear order $k \in [d]$, it holds that $i \leq_k j$ as otherwise $j \leq_L i$, and in at least one linear order $k' \in [d]$, it holds that $j \leq_{k'} i$ as otherwise $i \leq_L j$. Hence, $2 \leq l'_{ij} \leq 2d - 2 \leq (n+1) - 2 = n - 1 \leq l_{ij}$.

We can extend a tangle T' realizing L' such that we execute the remaining (even) number of $l_{ij} - l'_{ij}$ swaps of the wires i and j for each non-zero entry of L after an execution of an (i, j) swap in T'. Thus, the feasibility of L follows from the feasibility of L'. □

In the following we give an example of non-separable lists that are not feasible. Note that any triple $A \subseteq [n]$ of an even list is feasible if and only if it is non-separable (which is not true for general lists, e.g., the list $L = \{12, 23\}$ is not feasible).

Proposition 6 *There is an infinite family $(L^\star_m)_{m \geq 1}$ of non-separable lists whose entries are all zeros or twos such that $L^\star_m$ has 2^m wires and is not feasible for every $m \geq 4$.*

Slightly deviating from our standard notation, we number the wires of $L^\star_m$ from 0 to $2^m - 1$. There is no swap between two wires $i < j$ in $L^\star_m$ if each 1 in the binary representation of i also belongs to the binary representation of j, that is, the bitwise OR of i and j equals j; otherwise, there are two swaps between i and j. E.g., for $m = 4$, wire $1 = 0001_2$ swaps twice with wire $2 = 0010_2$, but doesn't swap with wire $3 = 0011_2$.

Each list $L^\star_m$ is clearly non-separable: assume that there exists a swap between two wires $i = (i_1 i_2 \ldots i_m)_2$ and $k = (k_1 k_2 \ldots k_m)_2$ with $k > i + 1$. Then there has to be some index a with $i_a = 1$ and $k_a = 0$. Consider any $j = (j_1 j_2 \ldots j_m)_2$ with $i < j < k$. By construction of $L^\star_m$, if $j_a = 0$, then there are two swaps between i and j; if $j_a = 1$, then there are two swaps between j and k.

We used two completely different computer programs[2] to verify that $L^\star_4$ – and hence every list $L^\star_m$ with $m \geq 4$ – is infeasible. Unfortunately, we could not find a combinatorial proof showing this. The list $L^\star_m$ has $\frac{1}{2}\sum_{r=1}^{m} 3^{r-1} 2^{m-r}(2^{m-r} - 1)$ swaps of multiplicity 2, so $L^\star_4$ has 55 distinct swaps. The full list $L^\star_4$ in matrix form is given below.

$$L^\star_4 = \begin{pmatrix}
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&2&0&2&0&2&0&2&0&2&0&2&0&2&0\\
0&0&0&0&2&2&0&0&2&2&0&0&2&2&0&0\\
0&0&0&0&2&2&2&0&2&2&2&0&2&2&2&0\\
0&0&0&0&0&0&0&0&2&2&2&2&0&0&0&0\\
0&0&0&0&0&0&2&0&2&2&2&2&2&0&2&0\\
0&0&0&0&0&0&0&0&2&2&2&2&2&2&0&0\\
0&0&0&0&0&0&0&0&2&2&2&2&2&2&2&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&2&0&2&0&2&0\\
0&0&0&0&0&0&0&0&0&0&0&0&2&2&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&2&2&2&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&2&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0
\end{pmatrix}$$

[2]One program is based on combining realizations for triplets of wires [11]; the other is based on a SAT formulation [1]. Both implementations are available on github.

5 Open Problems

Obviously it would be interesting to design faster algorithms for Tangle-Height Minimization and List-Feasibility. In particular, for the special case of simple lists, our exact algorithm running in $\mathcal{O}(n!\varphi^n)$ time and the algorithm of Baumann [2] running in $\mathcal{O}(n!\psi^n)$ time (where n is the number of wires, $\varphi \approx 1.618$, and $\psi \approx 1.325$) are not satisfactory given that odd-even sort [15] can compute a solution of height at most one more than the optimum in $\mathcal{O}(n^2)$ time. This leads to the question whether height-minimization is NP-hard for simple lists.

For general lists, one can potentially obtain a faster algorithm for List-Feasibility by improving the upper bound for entries of minimal feasible lists (see Proposition 2 for the current upper bound).

Let N be the number of permutations $\pi \in S_n$ such that there exists a simple consistent sublist L' of the given list L with $\mathrm{id}_n L' = \pi$. Clearly, $N \leq |S_n| = n!$. Better bounds for N could improve the runtime bounds stated in Sections 3.2 and 4.1.

Another research direction is to devise approximation algorithms for Tangle-Height Minimization.

In our setting the start permutation is given. If the task is to find a start permutation and a tangle that realizes the given list, new questions arise. Even in this setting, simple lists are not always feasible as we will show now. First note that for any permutation π in S_6, there exists a number $j_\pi \in \{1,2,3\}$ that, in π, lies between the other two numbers in the set $\{1,2,3\}$. Next, consider the list $L = \{41, 42, 51, 53, 62, 63\}$ of order 6. This list contains no swap among the wires in $\{1,2,3\}$, so in any realization, these never change their order. Moreover, for every ordering (i,j,k) of the wires in $\{1,2,3\}$, the list L contains a wire $m(j)$ in $\{4,5,6\}$ that swaps *only* with i and k (but not with j). For L to be realizable with start permutation π, however, wire $m(j_\pi)$ would need to swap also with wire j_π, which lies between the other two wires in $\{1,2,3\} \setminus \{j_\pi\}$. Can this new feasibility problem also be solved efficiently? What is the complexity of this problem for general lists?

Acknowledgments. We thank Thomas C. van Dijk for stimulating discussions in the initial phase of this work. We thank Stefan Felsner for discussions about the complexity of List-Feasibility. We thank the anonymous reviewers of the current version and earlier versions of this paper for helpful comments.

References

[1] Vasil Alistarov. Computing tangles using a SAT solver. Student report, 2022. Code available on github: https://github.com/alistairv/tangles-with-sat. URL: https://www1.pub.informatik.uni-wuerzburg.de/pub/theses/2022-alistarov-masterpraktikum.pdf.

[2] Jakob Baumann. Height minimization of simple tangles. Bachelor's thesis, September 2020. In German. URL: https://www.fim.uni-passau.de/fileadmin/dokumente/fakultaeten/fim/lehrstuhl/rutter/abschlussarbeiten/2020-Jakob_Baumann-BA.pdf.

[3] Sergey Bereg, Alexander Holroyd, Lev Nachmanson, and Sergey Pupyrev. Representing permutations with few moves. *SIAM J. Discrete Math.*, 30(4):1950–1977, 2016. URL: https://arxiv.org/abs/1508.03674, doi:10.1137/15M1036105.

[4] Sergey Bereg, Alexander E. Holroyd, Lev Nachmanson, and Sergey Pupyrev. Drawing permutations with few corners. In Stephen Wismath and Alexander Wolff, editors, *Proc. Int. Symp. Graph Drawing (GD'13)*, volume 8242 of *Lecture Notes Comput. Sci.*, pages 484–495. Springer, 2013. URL: http://arxiv.org/abs/1306.4048, doi:10.1007/978-3-319-03841-4_42.

[5] Joan S. Birman and R. Fredrick 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.

[6] James P. Crutchfield, J. Doyne Farmer, Norman H. Packard, and Robert S. Shaw. Chaos. *Scientific American*, 254(12):46–57, 1986. doi:10.1038/scientificamerican1286-46.

[7] Oksana Firman, Philipp Kindermann, Boris Klemz, Alexander Ravsky, Alexander Wolff, and Johannes Zink. The complexity of finding tangles. In Leszek Gąsieniec, editor, *Proc. 49th Int. Conf. Current Trends Theory & Practice Comput. Sci. (SOFSEM'23)*, volume 13878 of *Lecture Notes Comput. Sci.*, pages 3–17. Springer, 2023. doi:10.1007/978-3-031-23101-8_1.

[8] Oksana Firman, Philipp Kindermann, Alexander Ravsky, Alexander Wolff, and Johannes Zink. Computing optimal-height tangles faster. In Daniel Archambault and Csaba D. Tóth, editors, *Proc. 27th Int. Symp. Graph Drawing & Network Vis. (GD'19)*, volume 11904 of *Lecture Notes Comput. Sci.*, pages 203–215. Springer, 2019. URL: http://arxiv.org/abs/1901.06548, doi:10.1007/978-3-030-35802-0_16.

[9] Tosio Hiraguchi. On the dimension of orders. *The Science Reports of the Kanazawa University*, 4(1):1–20, 1955. doi:10.24517/00011473.

[10] Stuart A. Kauffman. *The Origins of Order: Self-Organization and Selection in Evolution*. Oxford University Press, 1993.

[11] Philipp Kindermann and Johannes Zink. Java and Python code for computing tangles. Github repository, 2022. URL: https://github.com/PhKindermann/chaotic-attractors.

[12] Donald E. Knuth. *The Art of Computer Programming, Volume III: Sorting and Searching*. Addison-Wesley, 2nd edition, 1998. URL: https://www.worldcat.org/oclc/312994415.

[13] Gabo Mindlin, Xin-Jun Hou, Robert Gilmore, Hernán Solari, and Nicholas B. Tufillaro. Classification of strange attractors by integers. *Phys. Rev. Lett.*, 64:2350–2353, 1990. doi:10.1103/PhysRevLett.64.2350.

[14] Maya Olszewski, Jeff Meder, Emmanuel Kieffer, Raphaël Bleuse, Martin Rosalie, Grégoire Danoy, and Pascal Bouvry. Visualizing the template of a chaotic attractor. In Therese Biedl and Andreas Kerren, editors, *Proc. 26th Int. Symp. Graph Drawing & Network Vis. (GD'18)*, volume 11282 of *Lecture Notes Comput. Sci.*, pages 106–119. Springer, 2018. URL: https://arxiv.org/abs/1807.11853, doi:10.1007/978-3-030-04414-5_8.

[15] Kazuhiro Sado and Yoshihide Igarashi. A function for evaluating the computing time of a bubbling system. *Theor. Comput. Sci.*, 54:315–324, 1987. doi:10.1016/0304-3975(87)90136-8.

[16] Thomas J. Schaefer. The complexity of satisfiability problems. In *Proc. 10th Annu. ACM Symp. Theory Comput. (STOC'78)*, pages 216–226, 1978. doi:10.1145/800133.804350.

[17] Ian Stewart. *Does God Play Dice? The New Mathematics of Chaos.* Wiley-Blackwell, 2nd edition, 2002.

[18] Paul Turán. On an extremal problem in graph theory. *Mat. Fiz. Lapok*, 48:436–452, 1941. In Hungarian.

[19] Deborah C. Wang. Novel routing schemes for IC layout part I: Two-layer channel routing. In *Proc. 28th ACM/IEEE Design Automation Conf. (DAC'91)*, pages 49–53, 1991. doi:10.1145/127601.127626.

[20] Wikipedia. Mathematical definition of attractors. https://en.wikipedia.org/wiki/Attractor#Mathematical_definition, 2003. Accessed April 10, 2025.

[21] Katsuhisa Yamanaka, Takashi Horiyama, Takeaki Uno, and Kunihiro Wasa. Ladder-lottery realization. In *Proc. 30th Canad. Conf. Comput. Geom. (CCCG'18)*, pages 61–67, 2018. URL: http://www.cs.umanitoba.ca/~cccg2018/papers/session2A-p3.pdf.