The Complexity of Finding Tangles

Oksana Firman¹, Philipp Kindermann², Boris Klemz¹, Alexander Ravsky³, Alexander Wolff¹, and Johannes Zink¹

¹ Institut für Informatik, Universität Würzburg, Würzburg, Germany
firstname.lastname@uni-wuerzburg.de
² Fachbereich IV – Informatikwissenschaften, Universität Trier, Germany
kindermann@uni-trier.de
³ Pidstryhach Institute for Applied Problems of Mechanics and Mathematics, National Academy of Sciences of Ukraine, Lviv, Ukraine
alexander.ravsky@uni-wuerzburg.de

Abstract. We study the following combinatorial problem. Given a set of n y-monotone curves, which we call wires, a tangle determines the order of the wires on a number of horizontal layers such that 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. Deciding whether a given multiset of swaps admits a realizing tangle is known to be NP-hard [Yamanaka et al., CCCG 2018]. We prove that this problem remains NP-hard if every pair of wires swaps only a constant number of times. On the positive side, we improve the runtime of a previous exponential-time algorithm. We also show that the problem is in NP and fixed-parameter tractable with respect to the number of wires.

Keywords: Tangle · NP-hard · Exponential-time algorithm · FPT

1 Introduction

This paper concerns the visualization of chaotic attractors, which occur in (chaotic) dynamic systems. Such systems are considered in physics, celestial mechanics, electronics, fractals theory, chemistry, biology, genetics, and population dynamics; see, for instance, [4], [13], and [6, p. 191]. Birman and Williams [3] were the first to mention tangles as a way to describe the topological structure of chaotic attractors. They investigated how the orbits of attractors are knotted. Later Mindlin et al. [9] characterized attractors using integer matrices that contain numbers of swaps between the orbits.

Olszewski et al. [10] studied the problem of visualizing chaotic attractors. Using \([n]\) as shorthand for \(\{1,2,\ldots,n\}\), define two permutations \(\sigma\) and \(\tau\) of \([n]\) to be adjacent if they differ only in transposing neighboring elements, that is, for every \(i \in [n]\), \(\sigma(i) \in \{\tau(i)\} \cup \{\tau(i-1) \mid i > 1\} \cup \{\tau(i+1) \mid i < n\}\). For two adjacent permutations \(\sigma\) and \(\tau\), let \(\text{diff}(\sigma, \tau) = \{\sigma(i), \sigma(i+1)\} \mid i \in [n-1] \land \sigma(i) = \tau(i+1) \land \sigma(i+1) = \tau(i)\}\) be the set of neighboring transpositions.
in which $\sigma$ and $\tau'$ differ. Given a set of $y$-monotone curves called wires that hang off a horizontal line in a prescribed order $\pi_0$, and a multiset $L$ (called list) of unordered pairs of wires (called swaps), the problem consists in finding a tangle realizing $L$, i.e., a sequence $\pi_0, \pi_1, \ldots, \pi_h$ of permutations of the wires such that (i) consecutive permutations are adjacent and (ii) $L = \bigcup_{i=0}^{h-1} \text{diff}(\pi_i, \pi_{i+1})$.

For example, the list $L$ in Fig. 1 admits a tangle realizing it. We call such a list feasible. The list $L' = L \cup \{(1, 2)\}$, in contrast, is not feasible. Note that, if the start permutation $\pi_0$ is not given explicitly, we assume that $\pi_0 = \text{id} = \langle 1, 2, \ldots, n \rangle$. In Fig. 2, the list $L_a$ is feasible; it is specified by an $(n \times n)$-matrix. The gray horizontal bars correspond to the permutations (or layers).

Olszewski et al. gave an exponential-time algorithm for minimizing the height of a tangle, that is, the number of layers. They tested their algorithm on a benchmark set, which showed that instances with up to 18 swaps can be solved within seconds, but instances with more than 22 swaps can take several hours.

We [5] showed, by reduction from 3-PARTITION, that tangle-height minimization is NP-hard. We also presented two (exponential-time) algorithms, one for the general problem and one for simple lists, that is, lists where each swap occurs at most once. Using an extended benchmark set, we showed that in almost all cases our algorithm for the general problem is faster and more memory-efficient than the algorithm of Olszewski et al.

In an independent line of research, Yamanaka et al. [16] 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.

Sado and Igarashi [11] used the same optimization criterion for tangles in the setting where only the beginning and final permutation are given (but they can choose the swaps performed to get there). They used odd-even sort, a parallel variant of bubble sort, to compute tangles with at most one layer more than the minimum in $O(n^2)$ time. Wang [15] showed that there is always a height-optimal tangle where no swap occurs more than once. Bereg et al. [1,2] 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).

### Notation and Conjecture.

For $n$ wires, a list $L = (l_{ij})$ of order $n$ is a symmetric $n \times n$ matrix with entries in $\mathbb{N}_0$ and zero diagonal. Let $|L| = \sum_{i<j} l_{ij}$ be the length of $L$. A list $L' = (l'_{ij})$ is a sublist of $L$ if $l'_{ij} \leq l_{ij}$ for each $i, j \in [n]$. If there is a pair $i, j \in [n]$ such that $l'_{ij} < l_{ij}$, then $L'$ is a strict sublist of $L$. A list is $0\cdot2$ if all entries are zeros or twos; it is even (odd) if all non-zero entries are even (odd).
The Complexity of Finding Tangles

For a list to be feasible, it also has to fulfill the following property. We say that a list is consistent if the final positions of all wires form a permutation of \([n]\). For a wire \(i\), its final position is its initial position (namely, \(i\)) minus one for each wire on its left that it swaps an odd number of times, plus one for each wire on its right that it swaps an odd number of times. We have shown that consistency is sufficient for the feasibility of odd lists [5]. Clearly, an even list is always consistent as, for any tangle realizing an even list, the initial permutation equals the final permutation.

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. It is natural to ask whether this necessary condition is also sufficient. For odd lists, non-separability is implied by consistency (because consistency is sufficient for feasibility [5]). Although the NP-hardness reduction from Section 3 shows that a non-separable list can fail to be feasible even when it is consistent. For even lists, the following question remains.

Conjecture 1 ([5]). Every non-separable even list is feasible.

In order to understood the structure of feasible lists better, we consider the following relation between them. Let \(L = (l_{ij})\) be a feasible list. Consider the list \(L'\) that is identical to \(L\) except that it has two additional \(i-j\) swaps. We claim that if \(l_{ij} > 0\) then the list \(L'\) is also feasible. Note that any tangle \(T\) that realizes \(L\) has a permutation \(\pi\) that supports the \(i-j\) swap. Directly after \(\pi\), 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 \(L \rightarrow L'\) if the list \(L\) can be extended to the list \(L'\) via the above operation.

For a list \(L = (l_{ij})\), let \(1(L) = (l_{ij} \mod 2)\) 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, \(L' \rightarrow L\) if and only if \(2(L') = 2(L)\) and \(l''_{ij} \leq l_{ij}\) for each \(i, j \in [n]\).
A feasible list $L_{\min}$ is minimal if there exists no feasible list $L^*$ such that $L^* \rightarrow L_{\min}$. Thus a list $L$ is feasible, if and only if there exists a minimal feasible list $L_{\min}$ of type 2($L$) such that $L_{\min} \rightarrow L$.

For a tangle $T$, let $L(T) = (l_{ij})$ be the symmetric $n \times n$ matrix with zero diagonal, where $l_{ij}$ is the number of $i$–$j$ swaps in $T$. Note that $T$ realizes $L(T)$.

Our Contribution. We call the problem of testing the feasibility of a given list List-Feasibility. As mentioned above, Yamanaka et al. [16] showed that this problem is NP-hard. 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 3. Our reduction uses a variant of Not-All-Equal 3-SAT (whereas Yamanaka et al. used 1-in-3 3SAT).

We start the paper, however, by studying exact algorithms for the List-Feasibility problem; see Section 2. We present an exponential-time algorithm with runtime $O((2|L|/n^2 + 1)^{n^2/2} \cdot n^3 \log |L|)$, where $L$ is the given list of order $n$. The runtime is expressed in terms of the logarithmic cost model of computation. This improves our previous algorithm [5] (which actually computes a tangle of minimum height, if possible). That algorithm runs in $O((2|L|/n^2 + 1)^{n^2/2} \cdot \varphi^n \cdot n)$ time in the unit-cost model and in $O((2|L|/n^2 + 1)^{n^2/2} \cdot \varphi^n \cdot n \log |L|)$ time in the log-cost model, where $\varphi \approx 1.618$ is the golden ratio. 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. As a corollary, this yields that List-Feasibility is in NP. Combined with our exponential-time algorithm, this also leads to a fixed-parameter tractable algorithm for testing feasibility (parameterized by the number of wires).

Finally, we disprove Conjecture 1; see Section 4. We could verify our counterexample (with 16 wires and 55 swaps of multiplicity 2) only by computer.

2 Exact Algorithms

We remind the reader of our algorithm for tangle-height minimization [5]: a dynamic program that runs in $O((2|L|/n^2 + 1)^{n^2/2} \cdot \varphi^n \cdot n)$ time in the unit-cost model; in the log-cost model the runtime increases by a factor of $O(\log |L|)$. We adjust this algorithm to the task of testing feasibility, which makes the algorithm simpler and faster. Then we will bound the entries of minimal feasible lists (defined above) 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 1.** There is an algorithm that, given a list $L$ of order $n$, tests whether $L$ is feasible in $O((2|L|/n^2 + 1)^{n^2/2} \cdot n^3 \cdot \log |L|)$ time in the log-cost model.

**Proof.** Let $F$ be a Boolean table with one entry for each sublist $L'$ of $L$ such that $F(L') = \text{true}$ 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 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 \( L \) of \( L \) is feasible if and only if there is a realizing tangle of \( L \) of height \(|L| + 1\). 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 \( F(L'') = \text{true} \), we compute the final positions of \( i \) and \( j \) with respect to \( L'' \). The desired tangle exists if and only if these positions differ by exactly one.

The number of sublists of \( L \) is upper bounded by \((2|L|/n^2 + 1)n^{7/2}\) [5]. For each sublist, we have to check \( O(n^2) \) swaps. To check a swap, we have to compute the final positions of two wires, which can be done in \( O(n \log |L|) \) time. \( \square \)

The following lemma follows from odd-even sort and is well-known [8].

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

Now we consider the following tangle shortening construction.

**Example 1.** Let \( T \) be a tangle and \( T' = (\pi_1, \ldots, \pi_k) \) be a subsequence of \( T \) containing the initial and the final permutations of \( T \). By Lemma 1, we can augment \( T' \) to a tangle \( T'' \), overwriting each two consecutive elements \( \pi_k \) and \( \pi_{k+1} \) of \( T'' \) by a tangle \( T_k'' \) which starts from \( \pi_k \), ends at \( \pi_{k+1} \), and whose list \( L(T_k'') = (l_{k,ij}) \) is simple. Now let \( T_k \) be the subtangle of \( T \) that starts from \( \pi_k \) and ends at \( \pi_{k+1} \). Let \( L(T_k) = (l_{k,ij}) \). The simplicity of the list \( L(T_k) \) implies that \( l_{k,ij} \leq l_{k,ij} \) for each \( i, j \in [n] \). It follows that \( l_{ij}' \leq \min\{1, l_{ij}, h - 1\} \) for each \( i, j \in [n] \), where \( L(T) = (l_{ij}) \) and \( L(T'') = (l_{ij}') \). Since the tangles \( T'' \) and \( T \) have common initial and final permutations, for each \( i, j \in [n] \) the numbers \( l_{ij} \) and \( l_{ij}' \) have the same parity, that is, \( 1(T'') = 1(T) \).

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 2 in Proposition 2 below.

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

**Proof.** The list \( L \) is feasible, so there is a tangle \( T \) realizing \( L \). We choose a subsequence \( T^* \) of \( T \) consisting of at most \( h = \left(\frac{n}{2}\right) + 2 \) permutations. To this end, we pick the initial and the final permutation of \( T \) and, for each pair \( (i, j) \in [n]^2 \) with \( i < j \) and \( l_{ij} \geq 1 \), we pick a permutation that swaps \( i \) and \( j \).

Let \( T' \) be the tangle that we construct from \( T \) using \( T^* \), as described in the shortening construction, and let \( L' = L(T') = (l_{ij}') \) be the list of \( T' \). The construction of the tangle \( T' \) assures that, for any \( i, j \in [n] \), if \( l_{ij} > 0 \), then \( l_{ij}' > 0 \). This, together with \( 1(L') = 1(L) \), yields \( 2(L') = 2(L) \). Hence, \( L' \rightarrow L \). List \( L \) is minimal, so \( L = L' \). Thus, \( l_{ij} = l_{ij}' \leq h - 1 \leq \left(\frac{n}{2}\right) + 1 \) for \( i, j \in [n] \). \( \square \)
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. Let \( T \) be a tangle (starting from id that realizes \( L \)). Given an \( i \to j \) swap, we define its span to be \(|i - j|\). Order the swaps in \( 1(L) \) according to decreasing span. We will color the swaps as follows. At each step we color in red the first non-colored swap \( i \to j \) (with \( i < j \)) from the list. Since the tangle \( T \) realizes the list \( L \), it contains a permutation \( \pi \) with \( \pi(j) < \pi(i) \). Put \( \pi \) into an initially empty set \( P \) for later use. Let \( k \in \{i + 1, \ldots, j - 1\} \) be an integer strictly between \( i \) and \( j \). Since \( \pi(j) < \pi(i) \), we have \( \pi(k) < \pi(i) \) or \( \pi(k) > \pi(j) \). We color in blue the \( i \to k \) swap in the former case and the \( k \to j \) swap in the latter case. In any case, if \( T_{\pi} \) is a tangle that starts from id and contains \( \pi \), the list \( L(T_{\pi}) \) contains the swap(s) we just colored in blue.

Let \( G \) be a graph with vertex set \([n]\) and edge set consisting of the red swaps. The coloring algorithm assures that the graph \( G \) is triangle-free, so, by Turán’s theorem [14], it has at most \( n^2/4 \) edges.

Let \( T^* \) be a subsequence of the tangle \( T \) consisting of the initial permutation id, all permutations in \( P \), and the final permutation of \( T \). Let \( T' \) be the tangle that we construct from \( T \) using \( T^* \) as described in Example 1. Let \( L' = L(T') \).

The construction of the tangle \( T' \) assures that if \( 1 \leq i < j \leq n \) and \( l_{ij} > 0 \) then \( \pi(j) < \pi(i) \) in some selected permutation \( \pi \). Since \( \pi \) is a member of the tangle \( T \), \( l'_{ij} > 0 \). Moreover, since \( 1(L') = 1(L) \), \( 2(L') = 2(L) \). Hence \( L' \to L \). The list \( L' \) is minimal, therefore \( L' = L \). Since each entry of the list \( L' \) is at most \(|P| + 2 - 1 \leq n^2/4 + 1 \), the same holds for the entries of the list \( L \). \( \square \)

Combining Proposition 2 and our exact algorithm yields a fixed-parameter tractable algorithm.

Theorem 2. 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 \( O((n/2)^n \cdot n^3 \log n + n^2 \log |L|) \) 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 Theorem 1 to check whether the list \( L' \) is feasible. Since our algorithm checks the feasibility of every 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 \( O(n^2 \log |L|) \) time. Note that \(|L'| \leq \left( \frac{n}{2} \right) \cdot (n^2/4 + 1) \leq (n^4 - 4n^2)/8 \). Plugging this into the runtime \( O((2L'|n^2 + 1)n^{5/2} \cdot n^3 \log |L'|) \) of our exact algorithm (Theorem 1) yields a total runtime of \( O((n/2)^n \cdot n^{5/2} \cdot n^3 \log n + n^2 \log |L|) \). \( \square \)

3 Complexity

Yamanaka et al. [16] 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. 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 [12], **Not-All-Equal 3-SAT** is NP-hard even if no negative 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 2.** **Positive NAE 3-SAT Diff** is NP-hard.

**Proof.** We show NP-hardness of **Positive NAE 3-SAT Diff** by reduction from **Not-All-Equal 3-SAT**. Let $\Phi = c_1 \land c_2 \land \cdots \land c_m$ be an instance of **Not-All-Equal 3-SAT** with variables $v_1, v_2, \ldots, v_n$. First we show how to get rid of negative variables and then of multiple occurrences of the same variable in a clause.

We create an instance $\Phi'$ 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 $(x_i \lor y_i \lor y_i)$. Now, we introduce three additional variables $a, b$, and $d$ that form the clause $(a \lor b \lor d)$. Let $c = (x \lor x \lor y)$ be a clause that contains two occurrences of the same variable. We replace $c$ by three clauses $(x \lor y \lor a), (x \lor y \lor b), (x \lor y \lor d)$. Since at least one of the variables $a, b, \text{or } d$ 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, $\Phi'$ is satisfiable if and only if $\Phi$ is. Clearly, the size of $\Phi'$ is polynomial in the size of $\Phi$. □

Our main result is as follows.

**Theorem 3.** **List-Feasibility** is NP-complete even if every pair of wires has at most eight swaps.

We split our proof into several parts. First, we introduce some notation, then we give the intuition behind our reduction. Next, we explain variable and clause gadgets in more detail. Finally, we show the correctness of the reduction.

**Notation.** We label the wires by their index in the initial permutation of a tangle. In particular, for a wire $\varepsilon$, its neighbor to the right is wire $\varepsilon + 1$. If a wire $\mu$ is to the left of some other wire $\nu$, then we write $\mu < \nu$. If all wires in a set $M$ are to the left of all wires in a set $N$, then we write $M < N$.

**Setup.** Given an instance $F = d_1 \land \cdots \land d_m$ of **Positive NAE 3-SAT Diff** with 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 $F$ is a yes-instance.

In $L$, we have two inner wires $\lambda$ and $\lambda' = \lambda + 1$ that swap eight times. This yields two types of loops (see Fig. 3): four $\lambda'$–$\lambda$ loops, where $\lambda'$ is on the left.
and \( \lambda \) is on the right side, and three \( \lambda' \)-loops with \( \lambda \) on the left and \( \lambda' \) on the right side. Notice that we consider only closed loops, which are bounded by swaps between \( \lambda \) and \( \lambda' \). 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 \( \lambda' \)-\( \lambda \) loops. We call the first two \( \lambda' \)-\( \lambda \) loops true-loops, and the last two \( \lambda' \)-\( \lambda \) 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 \( \lambda \) and \( \lambda' \), 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 \( F \). These sets are ordered (initially) \( V_n < V_{n-1} < \cdots < V_1 < \lambda < \lambda' < V'_1 < V'_2 < \cdots < V'_m \); 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 \( V' = V'_1 \cup V'_2 \cup \cdots \cup V'_m \). 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 \( D_{j_1}, D_{j_2}, \) and \( D_{j_3} \) that represent occurrences of variables in a clause \( d_j \) of \( F \). The wires in \( C_j \) are ordered \( D_{j_1} < D_{j_2} < D_{j_3} < c_j \). Together, the wires in \( C = C_1 \cup C_2 \cup \cdots \cup C_m \) represent the clause gadgets; they are ordered \( V < C_1 < C_2 < \cdots < C_m < \lambda \). Additionally, our list \( L \) contains a set \( E = \{ \varphi_1, \ldots, \varphi_\gamma \} \) 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. For each variable \( w_i \) of \( F \), \( i \in [n] \), we introduce two sets of wires \( V_i \) and \( V'_i \). Each \( V'_i \) contains a variable wire \( v_i \) that has four swaps with \( \lambda \) and no swaps with \( \lambda' \). Therefore, \( v_i \) intersects at least one and at most two \( \lambda' \)-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. 3. These wires neither swap with \( v_i \) nor with each other, but they have two swaps with both \( \lambda \) and \( \lambda' \). We want to force \( \alpha_i \) and \( \alpha'_i \) to have the two true-loops on their right and the two false-loops on their left, or vice versa. This will ensure that \( v_i \) cannot reach both a true- and a false-loop.

To this end, we introduce, for \( j \in [\gamma] \), a \( \beta_i \)-wire \( \beta_{i,j} \in V_i \) and a \( \beta'_i \)-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 \( \beta_i \)-wires as well as every pair of \( \beta'_i \)-wires swaps exactly once. Neither \( \beta_i \) nor \( \beta'_i \)-wires swap with \( \alpha_i \) or \( \alpha'_i \). Each \( \beta'_i \)-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 \( \lambda \) twice. Symmetrically, \( \beta_{i,2}, \beta_{i,4}, \beta'_{i,1}, \beta'_{i,3}, \beta'_{i,5} \) swap with \( \lambda' \) twice; see Fig. 3.

We use the \( \beta_i \) and \( \beta'_i \)-wires to fix the minimum number of \( \lambda' \)-\( \lambda \) loops that are on the left of \( \alpha_i \) and on the right of \( \alpha'_i \), respectively. Note that, together with \( \lambda \) and \( \lambda' \), the \( \beta_i \) and \( \beta'_i \)-wires have the same rigid structure as the wires shown in Fig. 2.

Observation 1 ([5]) The tangle in Fig. 2 realizes the list \( L_n \) specified there; all tangles that realize \( L_n \) have the same order of swaps along each wire.
This means that there is a unique order of swaps between the $\beta_i$-wires and $\lambda$ or $\lambda'$, 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 $\beta_i'$-wires. Hence, there are at least two $\lambda'-\lambda$ loops to the left of $\alpha_i$ and at least two to the right of $\alpha'_i$. Since $\alpha_i$ and $\alpha'_i$ do not swap, there cannot be a $\lambda'-\lambda$ loop that appears simultaneously on both sides.

Note that the $\lambda-\lambda'$ swaps that belong to the same side have to be consecutive, otherwise $\alpha_i$ or $\alpha'_i$ would need to swap more than twice with $\lambda$ and $\lambda'$. Thus, there are only two ways to order the swaps among the wires $\alpha_i$, $\alpha'_i$, $\lambda$, $\lambda'$; the order is either $\alpha_i'-\lambda'$, $\alpha'_i-\lambda$, four times $\lambda-\lambda'$, $\alpha_i'-\lambda'$, $\alpha'_i-\lambda$, $\alpha_i-\lambda'$, $\alpha_i'-\lambda$, $\alpha_i-\lambda$ (see Fig. 3(left)) or the reverse (see Fig. 3(right)). It is easy to see that in the first case $v_i$ can reach only the first two $\lambda'-\lambda$ 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 $j > i$, $\alpha_j$ and $\alpha'_j$ swap with both $\lambda$ and $\lambda'$ twice, the $\beta_i$-wires swap with $\alpha_i$ and $\alpha'_i$ twice, and, symmetrically, the $\beta'_i$-wires swap with $\alpha_i$ and $\alpha'_i$ twice, $v_j$ swaps with $\alpha_i$ and all wires in $V'_j$ six times. We briefly explain these multiplicities. Wires from $V_j$ and $V'_j \setminus \{v_j\}$ swap their partners twice so that they reach the corresponding $\lambda-\lambda'$ or $\lambda'-\lambda$ loops and go back. None of the wires from $V_i$ or $V'_i$ is restricted in which loop to intersect. Considering the wire $v_j$,
Fig. 4: 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.

Note that it has to reach the $\lambda' – \lambda$ loops twice. For simplicity and in order not to have any conflicts with the $\beta'_i$-wires, we introduce exactly six swaps with $\alpha_i$ and all wires in $V'_i$, see Fig. 4.

Clause gadget. For every clause $d_j$ from $F$, $j \in [m]$, we introduce a set of wires $C_j$. It contains the clause wire $c_j$ that has eight swaps with $\lambda'$. We want to force each $c_j$ to appear in all $\lambda' – \lambda$ loops. To this end, we use the set $E$ with the seven $\varphi$-wires $\varphi_1, \ldots, \varphi_7$ ordered $\varphi_1 < \cdots < \varphi_7$. They create a rigid structure similar to the one of the $\beta_i$-wires. Each pair of $\varphi$-wires swaps exactly once. For each $k \in [7]$, if $k$ is odd, then $\varphi_k$ swaps twice with $\lambda$ and twice with $c_j$ for every $j \in [m]$. If $k$ is even, then $\varphi_k$ swaps twice with $\lambda'$. Since $c_j$ does not swap with $\lambda$, each pair of swaps between $c_j$ and a $\varphi$-wire with odd index appears inside a $\lambda' – \lambda$ loop. Due to the rigid structure, each of these pairs of swaps occurs in a different $\lambda' – \lambda$ loop; see Fig. 5.

If a variable $w_i$ belongs to a clause $d_j$, then $L$ 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 $\lambda' – \lambda$ loops. This way, every clause contains at least one true and at least one false variable if $F$ is satisfiable.

We call a part of a clause wire $c_j$ that is inside a $\lambda' – \lambda$ loop—i.e., a $\lambda' – c_j$ loop—an arm of the clause $c_j$. 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 $\gamma^k$ 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 $\lambda'\cdots\lambda$ loop, $\gamma^k$ swaps twice with $\varphi_\ell$ for every odd $\ell \in [7]$.

Similarly to $c_j$, the wire $\gamma^k$ has eight swaps with $\lambda'$ and appears once in every $\lambda'\cdots\lambda$ loop. Additionally, $\gamma^k$ has two swaps with $c_j$.

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

The $\psi^k$-wires also have the rigid structure similar to the one that $\beta_i$-wires have, so that there is a unique order of swaps along each $\psi^k$-wire. Each pair of $\psi^k$-wires swaps exactly once, $\psi^k_{\lambda,1}$ and $\psi^k_{\lambda,3}$ have two swaps with $c_j$, and $\psi^k_{\lambda,2}$ has two swaps with $\lambda'$ and $v_i$. Note that no $\psi^k$-wire swaps with $\gamma^k$. Also, since $\psi^k_{\lambda,2}$ does not swap with $c_j$, the $\psi^k_{\lambda,2}$-$v_i$ swaps can appear only inside the $\lambda'$-$c_j$ loop that contains the arm of $c_j$ we want to protect from other variable wires. Since
Fig. 6: Tangle obtained from the satisfiable formula $F = (w_1 \lor w_2 \lor w_3) \land (w_1 \lor w_3 \lor w_4) \land (w_2 \lor w_3 \lor w_4) \land (w_2 \lor w_3 \lor w_5)$. Here, $w_1$, $w_3$, and $w_5$ are set to true, whereas $w_2$ and $w_4$ are set to false. We show only $\lambda$, $\lambda'$, and all variable and clause wires.

Inset: problems that occur if variable wires swap with clause wires in a different order.

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 $\gamma_j^k$ 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 $\gamma_j^k$; see Fig. 5.

Finally, we consider the behavior of wires from different clause gadgets among each other and with respect to wires from variable gadgets. For every $\ell > k$ and for every $j \in [m]$, the wires $c_j$ and $\gamma_j^\ell$ have eight swaps and the $\psi_j^\ell$-wires have two swaps with all wires in $C_j$. Since all wires in $V$ are to the left of all wires in $C$, each wire in $C$ swaps twice with all wires in $V$ and, for $i \in [n]$, with $\alpha_i'$. Finally, all $\alpha$- and $\alpha'$-wires swap twice with each $\varphi$-wire.

Note that the order of the arms of the clause wires inside a $\lambda'$–$\lambda$ 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. 6. 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 have to swap at least three times within the same $\lambda'$–$\lambda$ loop. However, we have reserved only eight swaps for each pair of variable wires—two for each of the four $\lambda'$–$\lambda$ loops.
Correctness. Clearly, if $F$ is satisfiable, then there is a tangle obtained from $F$ as described above that realizes the list $L$, so $L$ is feasible; see Fig. 6 for an example. On the other hand, if there is a tangle that realizes the list $L$ that we obtain from the reduction, then $F$ 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, a tangle exists if, for each clause, the corresponding three variable wires swap with the clause wire 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 $F$.

Membership in NP. To show that List-Feasibility is 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 $l'_{ij} \leq \min\{l_{ij}, n^2/4+1\}$ together with a permutation of its $O(n^4)$ swaps. Then we can efficiently test whether we can apply the swaps in this order to $id = (1, 2, \ldots, n)$. If yes, then the list $L'$ is feasible (and, due to $L' \rightarrow L$, a witness for the feasibility of $L$), otherwise we discard it. By Proposition 2, $L$ is feasible if and only if such a list $L'$ exists and is feasible.

4 Counterexample to Conjecture 1

Recall that Conjecture 1 claims that every non-separable even list is feasible. We showed that all non-separable 0–2 lists up to $n = 8$ wires are feasible [5].

We now construct a family $(L^*_m)_{m \geq 1}$ of non-separable 0–2 lists such that $L_m$ has $2^m$ wires and is not feasible for $m \geq 4$. We number the wires from 0 to $2^m - 1$. There is no swap between two wires $i < j$ in $L^*_m$ 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 = 0001₂ swaps twice with wire 2 = 0010₂, but doesn’t swap with wire 3 = 0011₂.

Each list $L^*_m$ is clearly non-separable: assume that there exists a swap between two wires $i = (i_1i_2\ldots i_m)_2$ and $k = (k_1k_2\ldots k_m)_2$ with $k > i+1$. Then there has to be some index $a$ with $i_a = 1$ and $k_a = 0$. Consider any $j = (j_1j_2\ldots j_m)_2$ with $i < j < k$. By construction of $L^*_m$, 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 confirmed by computer experiments (the Java source code is available on github [7]) that $L^*_4$ – and hence all $L^*_m$ with $m \geq 4$ – are infeasible. The list $L^*_m$ has $\frac{1}{2} \sum_{r=1}^{m} 3^{r-1} 2^{n-r} (2^{m-r} - 1)$ swaps of multiplicity 2, so $L^*_4$ has 55 distinct swaps. The full list $L^*_4$ in list form and in matrix form is given below. Unfortunately, we could not (yet) find a combinatorial proof that the non-separable 0–2 list $L^*_4$ is not feasible.

**Theorem 4.** Conjecture 1 is false.
Acknowledgments. We thank Stefan Felsner for discussions about the complexity of \textsc{List-Feasibility}.

$L^*_4 = 2\{(1,2), (1,4), (1,6), (1,8), (1,10), (1,12), (1,14), (2,4), (2,5), (2,8), (2,9), (2,12), (2,13), (3,4), (3,5), (3,6), (3,8), (3,9), (3,10), (3,12), (3,13), (3,14), (4,8), (4,9), (4,10), (4,11), (5,6), (5,8), (5,9), (5,10), (5,11), (5,12), (5,14), (6,8), (6,9), (6,10), (6,11), (6,12), (6,13), (7,8), (7,9), (7,10), (7,11), (7,12), (7,13), (7,14), (9,10), (9,12), (9,14), (10,12), (10,13), (11,12), (11,13), (11,14), (13,14)\}$

References

1. S. Bereg, A. Holroyd, L. Nachmanson, and S. Pupyrev. Representing permutations with few moves. \textit{SIAM J. Discrete Math.}, 30(4):1950–1977, 2016. URL: \url{https://arxiv.org/abs/1508.03674}, \url{doi:10.1137/15M1036105}.

2. S. Bereg, A. E. Holroyd, L. Nachmanson, and S. Pupyrev. Drawing permutations with few corners. In S. Wismath and A. Wolff, editors, \textit{Proc. Int. Symp. Graph Drawing (GD’13)}, volume 8242 of \textit{LNCS}, pages 484–495. Springer, 2013. URL: \url{http://arxiv.org/abs/1306.4048}, \url{doi:10.1007/978-3-319-03841-4_42}.

3. J. S. Birman and R. F. Williams. Knotted periodic orbits in dynamical systems—\textmd{I}: Lorenz’s equation. \textit{Topology}, 22(1):47–82, 1983. \url{doi:10.1016/0040-9383(83)90045-9}.

4. J. P. Crutchfield, J. D. Farmer, N. H. Packard, and R. S. Shaw. Chaos. \textit{Scientific American}, 254(12):46–57, 1986.

5. O. Firman, P. Kindermann, A. Ravsky, A. Wolff, and J. Zink. Computing optimal-height tangles faster. In D. Archambault and C. D. Tóth, editors, \textit{Proc. 27th Int. Symp. Graph Drawing & Network Vis. (GD’19)}, volume 11904 of \textit{LNCS},
6. S. A. Kauffman. *The Origins of Order: Self-Organization and Selection in Evolution*. Oxford University Press, 1993.

7. P. Kindermann and J. Zink. Java and Python code for computing tangles. Github repository, 2022. [URL: https://github.com/PhKindermann/chaotic-attractors](https://github.com/PhKindermann/chaotic-attractors).

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

9. G. Mindlin, X.-J. Hou, R. Gilmore, H. Solari, and N. B. Tufillaro. Classification of strange attractors by integers. *Phys. Rev. Lett.*, 64:2350–2353, 1990. [DOI: 10.1103/PhysRevLett.64.2350](https://doi.org/10.1103/PhysRevLett.64.2350).

10. 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, *Proc. 26th Int. Symp. Graph Drawing & Network Vis. (GD’18)*, volume 11282 of *LNCS*, pages 106–119. Springer, 2018. [URL: https://arxiv.org/abs/1807.11853](https://arxiv.org/abs/1807.11853), [DOI: 10.1007/978-3-030-04414-5_8](https://doi.org/10.1007/978-3-030-04414-5_8).

11. K. Sado and Y. 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](https://doi.org/10.1016/0304-3975(87)90136-8).

12. T. 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](https://doi.org/10.1145/800133.804350).

13. I. Stewart. *Does God Play Dice?: The New Mathematics of Chaos*. Penguin, 1997.

14. P. Turán. On an external problem in graph theory. *Mat. Fiz. Lapok*, 48:436–452, 1941.

15. D. 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](https://doi.org/10.1145/127601.127626).

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