The Complexity of Finding Tangles

Oksana Firman[1], Stefan Felsner[2], Philipp Kindermann[1], Alexander Ravsky[3], Alexander Wolff[1], and Johannes Zink[1]

1 Universität Würzburg, Germany
`firstname.lastname@uni-wuerzburg.de`

2 TU Berlin, Germany
`felsner@math.tu-berlin.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 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. Finding a tangle that realizes a given multiset of swaps and uses the least number of layers is known to be NP-hard. We show that it is even NP-hard to decide if a realizing tangle exists.

1 Introduction

The subject of this paper is the visualization of so-called *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. 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. [6] showed how to characterize attractors using integer matrices that contain numbers of swaps between the orbits.

Olszewski et al. [7] studied computational aspects of visualizing chaotic attractors. In the framework of their paper, one is given a set of y-monotone curves called *wires* that hang off a horizontal line in a fixed order, and a multiset of swaps between the wires (called *list*). A *tangle* then is a visualization of these swaps, i.e., a sequence of permutations of the wires such that consecutive permutations differ only in swaps of neighboring wires (but disjoint swaps can be done simultaneously). For examples of lists and tangles realizing them, see Figs. 1 and 2. The list L in Fig. 1 admits a tangle realizing it. We call such a list *feasible*. The list L', in contrast, is not feasible. In Fig. 2, the list L_n is described by an $(n \times n)$-matrix. The gray horizontal bars correspond to the permutations (or *layers*). Olszewski et al. gave

$$L = \{(1,2),(1,3)\}$$

$$L' = \{(1,2)_2,(1,3)\}$$

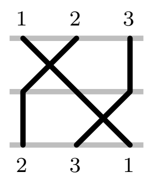

Figure 1 Lists L and L' for three wires (left). The list L is feasible (a tangle realizing L is to the right), whereas L', which has two copies of the swap $(1,2)$, is infeasible.

36th European Workshop on Computational Geometry, Würzburg, Germany, March 16–18, 2020.
This is an extended abstract of a presentation given at EuroCG'20. It has been made public for the benefit of the community and should be considered a preprint rather than a formally reviewed paper. Thus, this work is expected to appear eventually in more final form at a conference with formal proceedings and/or in a journal.

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

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

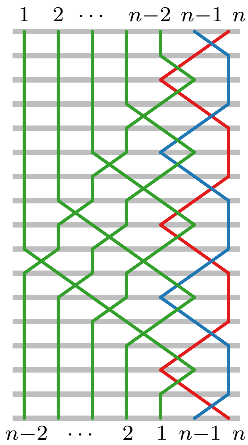

Figure 2 A list L_n for n wires (left) and a tangle realizing L_n (right). Entry (i, j) of L_n defines how often wires i and j must swap in the tangle. Here, $n = 7$.

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.

Later, we [5] showed that in fact tangle-height minimization is NP-hard. Our proof was by reduction from 3-Partition. We also presented an (exponential-time) algorithm for the problem. Using an extended benchmark set, we showed that in almost all cases our algorithm is faster than the algorithm of Olszewski et al.

Sado and Igarashi [8] used the same optimization criterion for tangles, given only the final permutation. They used odd-even sort, a parallel variant of bubble sort, to compute tangles with at most one layer more than the minimum. Wang [10] 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).

In this paper we strengthen our previous results and show that it is even NP-hard to test, given a multiset of swaps and a start permutation of the wires, whether there is *any* tangle that realizes the given swaps. We call this problem List-Feasibility.

2 Complexity

We show that List-Feasibility is NP-hard by reducing 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 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 [9], 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. It is easy to see that this variant is NP-hard, too.

▶ **Lemma 1.** Positive NAE 3-SAT Diff *is NP-hard.*

For a formal proof, see the full version [4]. Our main result is as follows.

▶ **Theorem 2.** List-Feasibility *is NP-hard (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 ε, its neighbor to the right is wire $\varepsilon + 1$. If a wire μ is to the left of some other wire ν, we write $\mu < \nu$. If all wires in a set M are to the left of all wires in a set N, we write $M < N$. For any integer $k > 0$, let $[k] = \{1, 2, \dots, k\}$.

Setup. Given an instance $F = d_1 \wedge \dots \wedge d_m$ of POSITIVE NAE 3-SAT DIFF with variables $w_1, \dots, w_n$, 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 λ and $\lambda' = \lambda + 1$ that swap eight times. This yields two types of loops (see Fig. 3): four λ'–λ loops, where λ' is on the left and λ is on the right side, and three λ–λ' loops with λ on the left and λ' on the right side. Notice that we consider only *closed* loops, which are bounded by swaps between λ and λ'. 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 F. These sets are ordered (initially) $V_n < V_{n-1} < \dots < V_1 < \lambda < \lambda' < V'_1 < V'_2 < \dots < V'_n$; the order of the wires inside these sets will be detailed in the next two paragraphs. Let $V = V_1 \cup V_2 \cup \dots \cup V_n$ and $V' = V'_1 \cup V'_2 \cup \dots \cup V'_n$. 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_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 $D_j^3 < D_j^2 < D_j^1 < c_j$. Together, the wires in $C = C_1 \cup C_2 \cup \dots \cup C_m$ represent the clause gadgets; they are ordered $V < C_m < C_{m-1} < \dots < C_1 < \lambda$. Additionally, our list L contains a set $E = \{\varphi_1, \dots, \varphi_7\}$ 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. 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 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. 3. 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 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 $j \in [5]$, a β_i*-wire* $\beta_{i,j} \in V_i$ and a β'_i*-wire* $\beta'_{i,j} \in V'_i$. These are ordered $\beta_{i,5} < \beta_{i,4} < \dots < \beta_{i,1} < \alpha_i$ and $\alpha'_i < \beta'_{i,1} < \beta'_{i,2} < \dots < \beta'_{i,5} < v_i$. 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. 3.

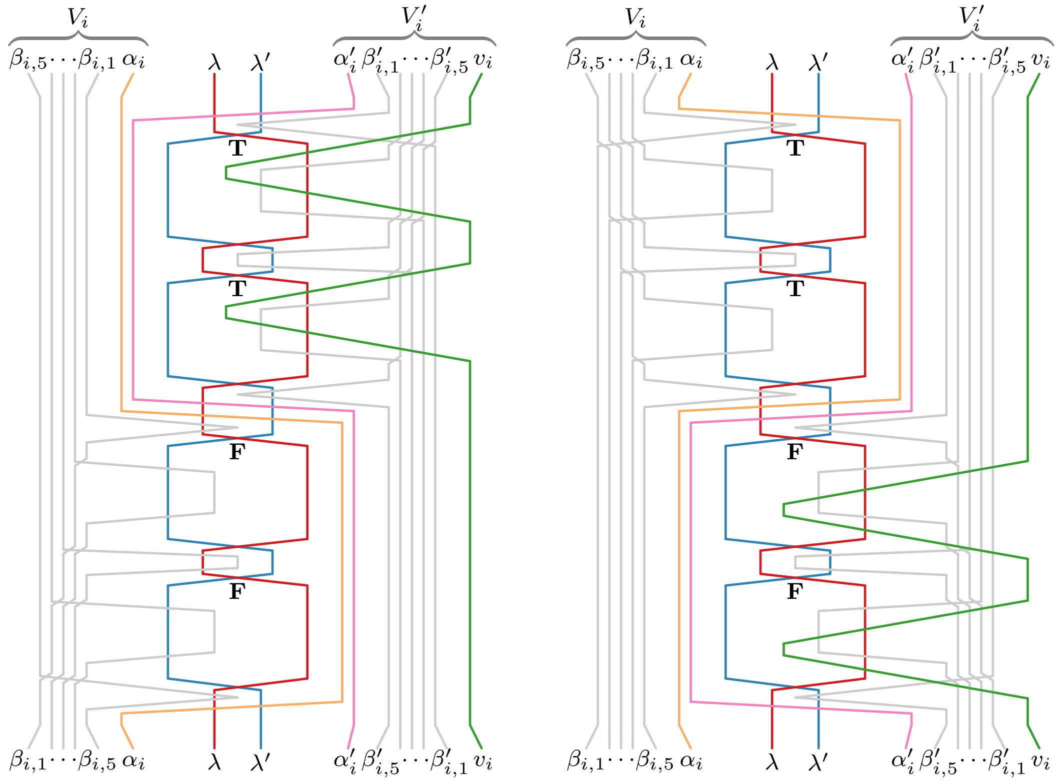

Figure 3 A variable gadget with a variable wire v_i that corresponds to the variable that is true (on the left) or false (on the right).

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 rigid structure as the wires 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 β_i-wires and λ or λ', i.e., for $j \in [4]$, every pair of $\beta_{i,j+1}$–λ swaps (or $\beta_{i,j+1}$–λ' swaps, depending on the parity of j) can be done only after the pair of $\beta_{i,j}$–λ' swaps (or $\beta_{i,j}$–λ 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 to the left of α_i and at least two 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 λ–λ' swaps that belong to the same side have to be consecutive, otherwise α_i or α'_i would need 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', α_i–λ, α_i–λ', four times λ–λ', α_i–λ', α_i–λ (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 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 twice, and, symmetrically, the β'_j-wires swap with α_i and V'_i twice, v_j swaps with α_i and all wires in V'_i six times. We

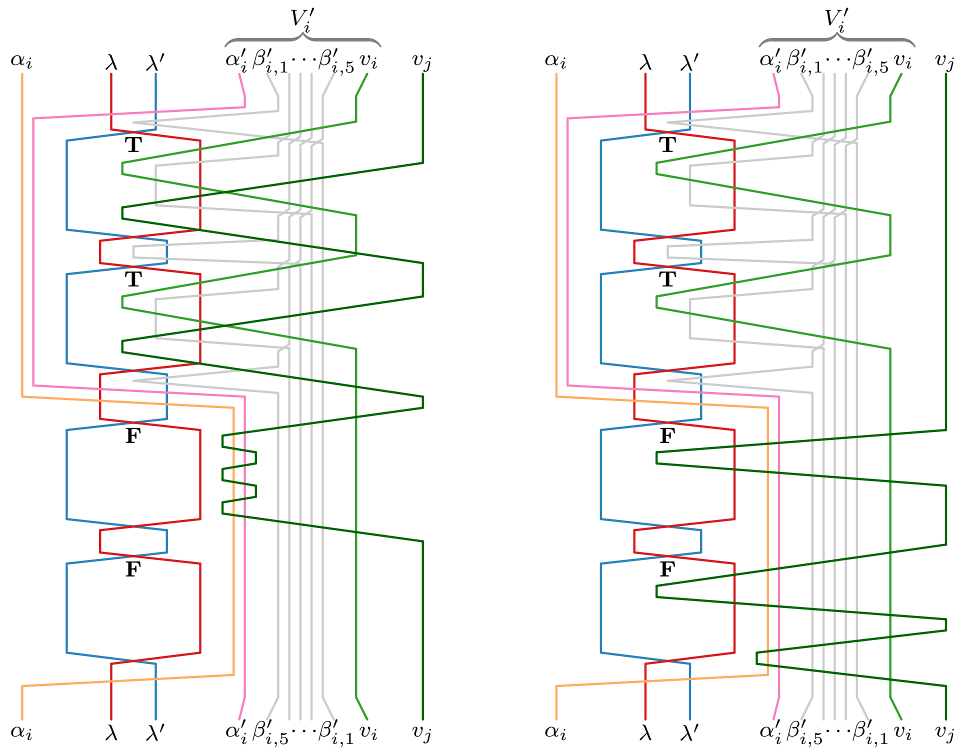

Figure 4 A realization of swaps between the variable wire v_j and all wires that belong to the variable gadget corresponding to the variable w_i. On the left the variables w_i and w_j are both true, and on the right w_i is true, whereas w_j is false.

briefly explain these multiplicities. Wires from V_j and $V_j' \setminus \{v_j\}$ swap their partners twice so that they reach the corresponding λ–λ' or λ'–λ 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, note that it has to reach the λ'–λ loops twice. For simplicity and in order not to have any conflicts with the β_i'-wires, we introduce exactly six swaps with α_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 λ'. We want to force each c_j to appear in all λ'–λ loops. To this end, we use the set E with the seven *φ-wires* $\varphi_1, \ldots, \varphi_7$ ordered $\varphi_1 < \cdots < \varphi_7$. They create a rigid structure similar to the one of the β_i-wires. Each pair of φ-wires swaps exactly once. For each $k \in [7]$, if k is odd, φ_k swaps twice with λ and twice with c_j for every $j \in [m]$. If k is even, 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. 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 λ'–λ 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 λ'–λ loop—i.e., a λ'–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 γ_j^k will protect the arm of c_j that the variable wire of the k-th variable

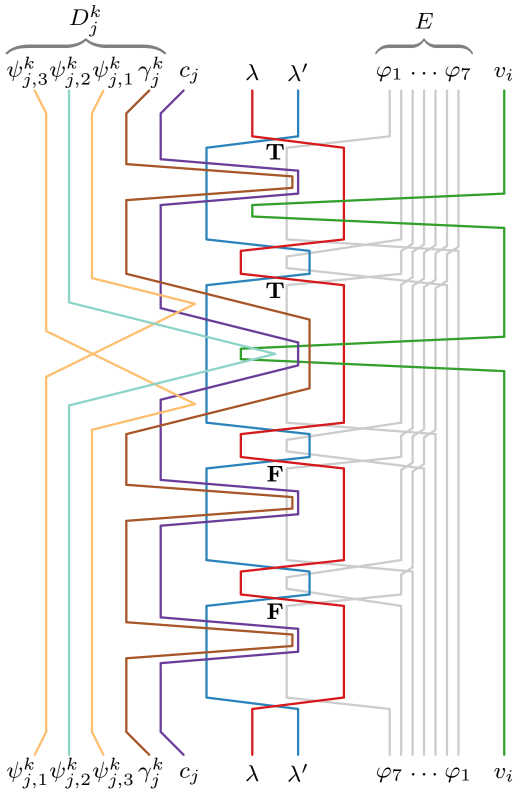

Figure 5 A gadget for clause c_j showing only one of the three variables, namely v_i.

of d_j intersects. To achieve this, γ_j^k has to intersect one of the φ-wires that swaps with the arm. In order not to restrict the choice of the λ'–λ loop, γ_j^k swaps twice with each φ_ℓ with 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.

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 $\psi_{j,3}^k < \psi_{j,2}^k < \psi_{j,1}^k < \gamma_j^k$.

The ψ_j^k-wires also have the rigid structure similar to the one that β_i-wires have, so that there is a unique order of swaps along each ψ_j^k-wire. 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. Note that 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 $\psi_{j,2}^k$–λ' 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. 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 γ_j^ℓ have eight swaps and the ψ_j^ℓ-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 α_i'. Finally, all α- and α'-wires swap twice with each φ-wire.

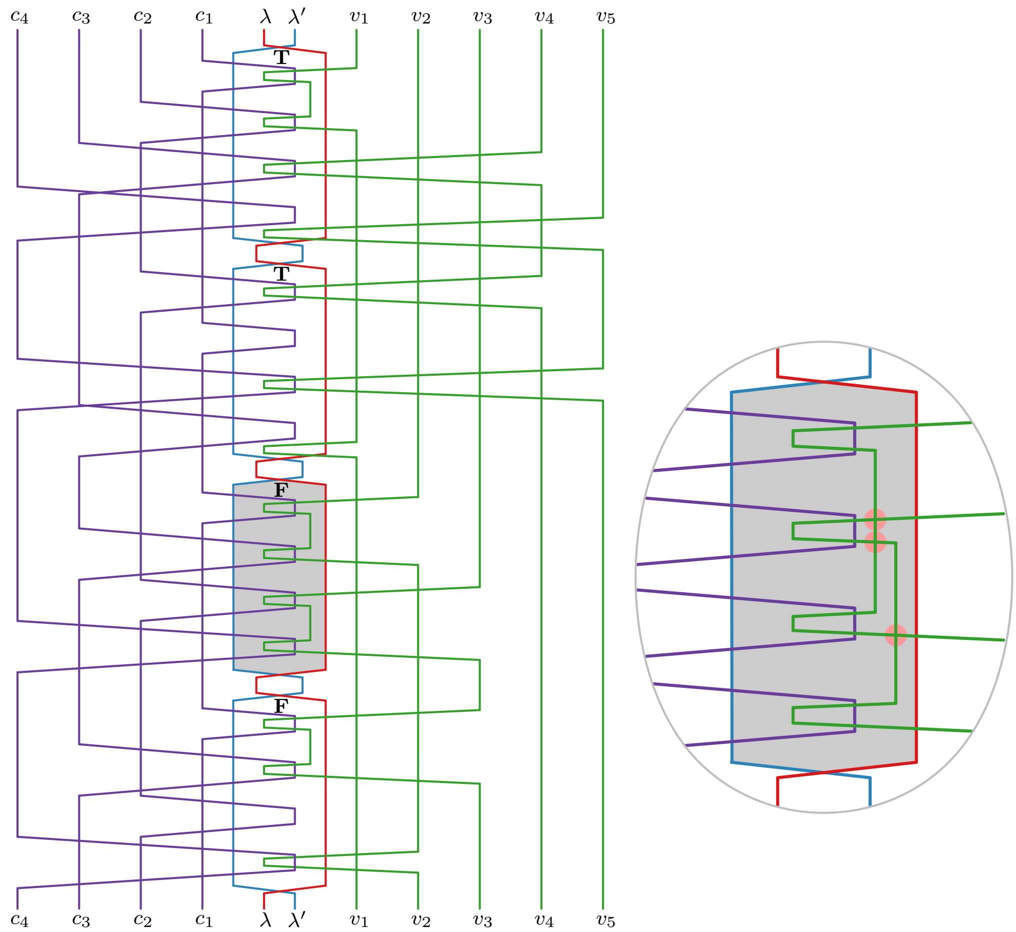

Figure 6 A tangle obtained from the satisfiable formula $F = (w_1 \vee w_2 \vee w_3) \wedge (w_1 \vee w_3 \vee w_4) \wedge (w_2 \vee w_3 \vee w_4) \wedge (w_2 \vee w_3 \vee w_5)$. Here w_1, w_4 and w_5 are set to true, whereas w_2 and w_3 are set to false. Note that we show here only "crucial" wires, namely λ, λ', and all variable and clause wires.

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 should be consecutive, 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 λ'–λ loop. However, we have reserved only eight swaps for each pair of variable wires—two for each of the four λ'–λ 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.

3 Open Problems

We recall three open questions of our previous paper [5].

- Can we decide the feasibility of a list faster than finding its optimal realization?
- For *simple* lists, that is, lists where each swap occurs at most once, odd-even sort efficently computes tangles with at most one layer more than minimum [8]. Can minimum-height tangles be computed efficiently for simple lists?
- In this paper, we showed that it is NP-hard to decide whether general lists are feasible. Earlier, we showed that the problem is easy for odd lists [5], i.e., if every swap occurs an odd number of times or zero times. A list is *non-separable* if, for every three wires $i < k < j$, there is no swap between i and k, and none between k and j, then there isn't any between i and j either. We conjecture that every non-separable even list is feasible.

References

1. 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`.
2. 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 *LNCS*, pages 484–495. Springer, 2013. URL: `http://arxiv.org/abs/1306.4048`, `doi:10.1007/978-3-319-03841-4_42`.
3. 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`.
4. Oksana Firman, Stefan Felsner, Philipp Kindermann, Alexander Ravsky, Alexander Wolff, and Johannes Zink. The complexity of finding tangles. Arxiv report, 2020. URL: `http://arxiv.org/abs/2002.12251`.
5. 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 *LNCS*, pages 203–215. Springer, 2019. URL: `http://arxiv.org/abs/1901.06548`, `doi:10.1007/978-3-030-35802-0_16`.
6. Gabo Mindlin, Xin-Jun Hou, Robert Gilmore, Hernán Solari, and N. B. Tufillaro. Classification of strange attractors by integers. *Phys. Rev. Lett.*, 64:2350–2353, 1990. `doi:10.1103/PhysRevLett.64.2350`.
7. 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 *LNCS*, pages 106–119. Springer, 2018. URL: `https://arxiv.org/abs/1807.11853`, `doi:10.1007/978-3-030-04414-5_8`.
8. 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`.
9. 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`.
10. 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`.