Secondary Structure Design for Cotranscriptional 3D RNA Origami Wireframes

Pekka Orponen
Department of Computer Science, Aalto University, Finland

Shinnosuke Seki
Department of Computer and Network Engineering, University of Electro-Communications, Japan

Antti Elonen
Department of Computer Science, Aalto University, Finland

Abstract

We address the task of secondary structure design for *de novo* 3D RNA origami wireframe structures in a way that takes into account the specifics of a cotranscriptional folding setting. We consider two issues: firstly, avoiding the topological obstacle of "polymerase trapping", where some helical domain cannot be hybridised due to a closed kissing-loop pair blocking the winding of the strand relative to the polymerase–DNA-template complex; and secondly, minimising the number of distinct kissing-loop designs needed, by reusing KL pairs that have already been hybridised in the folding process. For the first task, we present an efficient strand-routing method that guarantees the absence of polymerase traps for *any* 3D wireframe model, and for the second task, we provide a graph-theoretic formulation of the minimisation problem, show that it is NP-complete in the general case, and outline a branch-and-bound type enumerative approach to solving it. Both algorithms have been implemented in the *DNAforge* design tool (`https://dnaforge.org`) and we present some examples of the results.

2012 ACM Subject Classification Theory of computation → Theory and algorithms for application domains; Applied computing → Life and medical sciences → Computational biology

Keywords and phrases RNA origami, wireframe nanostructures, cotranscriptional folding, secondary structure, kissing loops, algorithms, self-assembly

Digital Object Identifier 10.4230/LIPIcs.DNA.2025.

31st International Conference on DNA Computing and Molecular Programming (DNA31).
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

Concurrently to the advances in DNA nanotechnology, there has been increasing interest in using RNA as the fabrication material for self-assembling bionanostructures. In comparison to DNA, the appeal of RNA is that the strands can be produced by the natural process of polymerase transcription, and the structures can thus be created in room temperature *in vitro*, and possibly eventually *in vivo*, from genetically engineered DNA templates. The challenge, on the other hand, is that the folding process of RNA is kinetically more complex and hence less predictable than DNA helix formation, at least at the present stage of RNA engineering.

The first design technique applied in this area of *RNA nanotechnology* was modular "RNA tectonics", in which naturally occurring RNA structures are connected together to create bigger target complexes using specific connector motifs such as sticky-end pairings and a variety of pseudoknots [10, 11]. A complementary top-down method of "RNA origami", in which a task-specific strand is rationally designed to fold into the desired target structure, was then introduced in a pioneering work by Geary et al. in 2014 [7]. Geary et al. demonstrated the feasibility of their method by synthesising 2D "RNA tiles" of different sizes, and this approach has since then been further developed with new design motifs, techniques, and tools [13, 5]. (For an overview, see [16].)

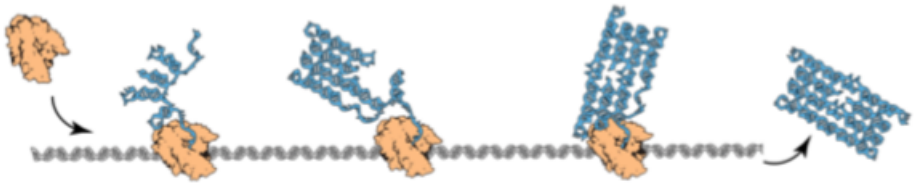

Figure 1 Cotranscriptional folding of a 2D RNA origami tile structure from a DNA template, mediated by an RNA polymerase enzyme. (Reprinted with permission from [6].)

One emphasis in the work of Geary et al. [7, 5] has been the *cotranscriptional* nature of the polymerase transcription process, that is, the way the transcribed RNA strand starts to fold into secondary structures already while being spooled off the polymerase enzyme (Figure 1). This characteristic of natural RNA generation introduces new challenges and also opportunities for the rational design process, some of which we shall explore in the present work.

In the following, Section 2 presents a spanning-tree based framework for self-assembly of wireframe structures by co-transcriptional folding, and introduces the topological folding obstacle of polymerase trapping. Section 3 then demonstrates how this obstacle can always be avoided by using a depth-first-search (DFS) based design scheme. Section 4 introduces the notion of the KLX number of a DFS tree. This corresponds to the maximum number of kissing loops that are simultaneously "open" in the folding process, and hence need different designs in order to avoid nonspecific pairings. Minimising this number provides the possibility of reusing KL designs. However, as proved in Section 5, the KLX minimisation problem is in the general case NP-hard. Section 6 thus provides a branch-and-bound type enumeration algorithm for the problem. Section 7 provides some examples of using the *DNAforge* tool to compute the DFS tree based designs and KLX minimisation. Section 8 summarises the results and suggests some directions for further work.

2 Wireframe RNA origami and the polymerase trapping obstacle

An extension of the RNA origami method to the design of 3D wireframe structures was presented by Elonen et al. in [3]. We conduct our discussion in this framework, but the basic ideas apply, *mutatis mutandis*, also to the task of designing 2D RNA origami tiles (cf. [14]). The general spanning-tree based 3D wireframe design scheme is outlined in Figure 2.

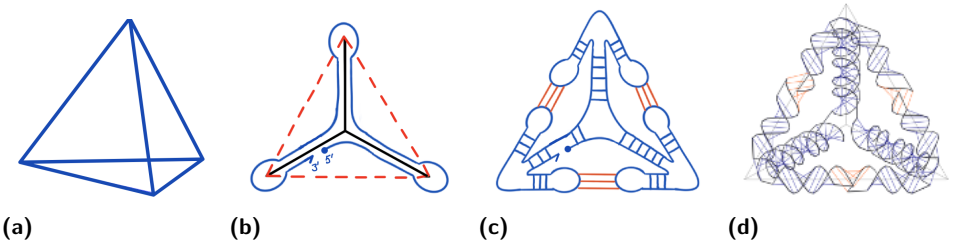

Figure 2 A spanning-tree based design scheme for 3D RNA wireframe origami. (a) Targeted wireframe model. (b) A spanning tree and strand routing of the wireframe graph. (c) Routing-based stem and kissing-loop pairings. (d) Helix-level model. (Adapted with permission from [3].)

In this scheme, one starts from the targeted wireframe, which in the case of Figure 2(a) is a simple tetrahedron. (Or more precisely the wireframe skeleton of a tetrahedral mesh.) In the first design step (Figure 2(b)) one chooses some spanning tree T of the wireframe graph G,[1] and designs the primary structure of the RNA strand so that it folds to create a twice-around-the-tree walk on T, covering each edge of T twice in antiparallel directions. In the second design step (Figure 2(c)) one then extends the walk halfway along each of the co-tree (= non-spanning tree) edges of G into a hairpin loop, and designs the base sequences at the termini of the hairpins so that pairwise matching half-edges are connected by the 180° kissing-loop design motif introduced in [7], thus constituting the co-tree edges. Figure 2(d) presents a helix-level model of the eventual nanostructure.

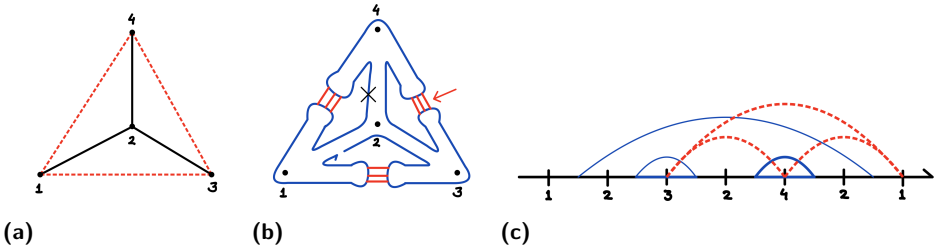

Figure 3 A tetrahedron design based on a 3-star spanning tree. (a) Spanning tree and co-tree of the tetrahedral graph. (b) Strand routing and kissing-loop pairs for the design. (c) Domain-level arc diagram of the design.

One potentially significant topological obstacle to cotranscriptional folding in this framework is the phenomenon of *polymerase trapping*, identified by Geary and Andersen in [8] and analysed by Mohammed et al. in [14].

To explain this concern, let us review the previous tetrahedron design, presented in more detail in Figure 3. Figure 3(a) shows the tetrahedral wireframe as a Schlegel diagram, that

[1] A spanning tree of a graph G is an acyclic subgraph that connects all the vertices of G.

is, as a planar projection from a point above one of the tetrahedron's faces. The edges of the chosen spanning tree, which in this case is a 3-pointed star, are indicated by solid black lines, and the co-tree edges by dashed red lines.

Figure 3(b) depicts again the corresponding twice-around-the-tree strand routing (blue) and the complementary kissing-loop pairings (red). The helix junctions in the design, which constitute the vertices of the eventual 3D nanostructure, are now indexed according to their first-visit order in the strand routing.

The schematic in Figure 3(c) presents the design as a domain-level arc diagram, where the strand is laid out along a line in the 5′ to 3′ direction, the vertex visits are marked by the corresponding indices, the domain-to-domain helical pairings are indicated by solid blue arcs, and the kissing-loop pairings by red dashed arcs. (For simplicity and clarity, the half-edge stem pairings flanking each kissing-loop hairpin are not presented.)

Consider now how a cotranscriptional folding process of this structure might proceed. Instead of thinking of the RNA strand being spooled out of the polymerase starting at the 5′ end and folding as the appropriate base pairings become available, it may be easier to visualise the large polymerase–DNA-template complex as traversing the 5′-3′ strand route outlined in Figure 3(c) and transcribing the nucleotide domains as it goes.

First the domains 1-2 and 2-3 are transcribed, and the RNA strand stays linear until the transcription of domain 3-2 begins. (For simplicity, we are ignoring any transient nonspecific nucleotide pairings that arise during the folding process.) Between the completion of domain 2-3 and the initiation of domain 3-2, the two opening hairpins for the kissing loops 3-4 and 3-1 are transcribed. (The best relative ordering of these two transcriptions is a geometric and sequence-design issue, and we leave this choice open in this high-level view.) After (or while) the complementary domains 2-3 and 3-2 hybridise, domain 2-4 is transcribed, and after that the closing hairpin of the 3-4 kissing loop and the opening hairpin of the 4-1 kissing loop, in some order.

Consider now what happens when the polymerase reaches domain 4-2 (marked with a black cross in diagram 3(b)), where it should create a double-stranded helix with domain 2-4, by winding the strand around it in antiparallel direction. If the 3-4 kissing loop (marked with a red arrow in 3(b)) has already closed, the strand with the big polymerase–DNA complex coupled to it cannot achieve this, since the kissing-loop pairing is blocking the pathway. (This is of course also a time-scale issue, and depends among other things on the strand distance between the closing hairpin of the kissing loop and the closing domain of the helical pairing; but let us again ignore these lower-level details at this presentation.)

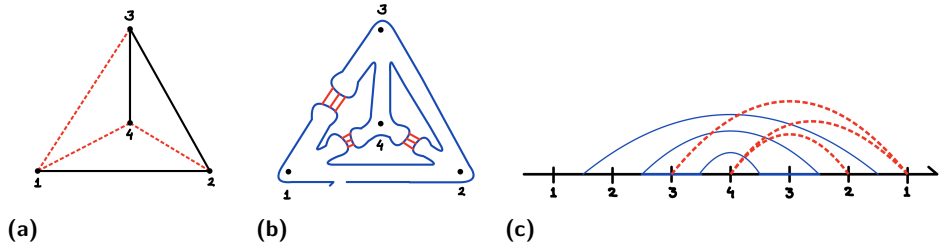

Figure 4 A tetrahedron design based on a 4-path spanning tree. (a) Spanning tree and co-tree of the tetrahedral graph. (b) Strand routing and kissing-loop pairs for the design. (c) Domain-level arc diagram of the design.

Schematically, one can see that the risk of this kind of "polymerase trapping" topological obstacle emerges when a kissing-loop pair that has been initiated (opened) before a given

helical pairing closes between the opening and closing of the helical pairing; or in terms of our arc diagram when a "red arc" that has been initiated before a "blue arc" closes inside that blue arc.

As another example, let us consider the tetrahedron design presented in Figure 4. As shown in Figure 4(a), in this case the spanning tree is a simple 4-vertex path. Figure 4(b) again outlines the corresponding strand route and kissing-loop pair arrangement, with the helix junctions numbered according to their first-visit order. As witnessed by Figure 4(c), this time there is no risk for the polymerase trapping obstacle. That is, every kissing loop closes only after the completion of all the helical pairings that were open when the kissing-loop pair was initiated.

Such complete absence of polymerase traps seems like a very particular property, and one may wonder for which kinds of wireframe models this condition can be achieved. As we shall see in the next section, however, such an arrangement of the helical and kissing loop pairings can in fact be found for *any* connected wireframe graph, by an application of the fundamental algorithmic method of depth-first search [1, Sec. 20.3].

3 Cotranscription-friendly secondary structure design

Algorithm 1 Depth-first search of a graph $G = (V, E)$ from root vertex $r \in V$

```
Initially all vertices v ∈ V and edges e ∈ E are set to be unmarked.

function DFS(G, r)
    mark vertex r as visited
    for each edge e = {r, v} incident to r do
        if vertex v is not marked as visited then
            mark e as a tree edge
            perform search DFS(G, v)
        else
            mark e as a back edge, unless it is already marked (= edge to parent)
        end if
    end for
end function
```

To streamline the presentation, we assume henceforth that any graph under consideration is connected and undirected, unless stated otherwise. The *depth-first search (DFS)* method for systematically traversing and labelling a (connected, undirected) graph is presented as Algorithm 1.

Consider a graph $G = (V, E)$ with $|V| = n$ vertices and $|E| = m$ edges. Then a DFS traversal of G, starting from any chosen *root* vertex $r \in V$, partitions the set of edges E in time $O(m)$ in two disjoint classes: $n - 1$ *tree edges* and $m - n + 1$ *back edges*. The tree edges constitute a spanning tree T of G, which can be considered to be rooted at r and oriented accordingly, and the back edges (which constitute the corresponding co-tree $E \setminus T$) have the important property that they can always be oriented to point "upward" towards the root of the tree, that is, there are no "cross edges" connecting two different branches of the directed tree. (For examples, see Figures 5 and 6.)

To make this precise, and to introduce another important notion, consider such a *DFS (spanning) tree* for a graph $G = (V, E)$ to be a four-tuple $T = (V, S, r, \delta)$, where $S \subseteq E$ is the set of tree or *stem* edges, $r \in V$ is the chosen root which determines the orientation of the

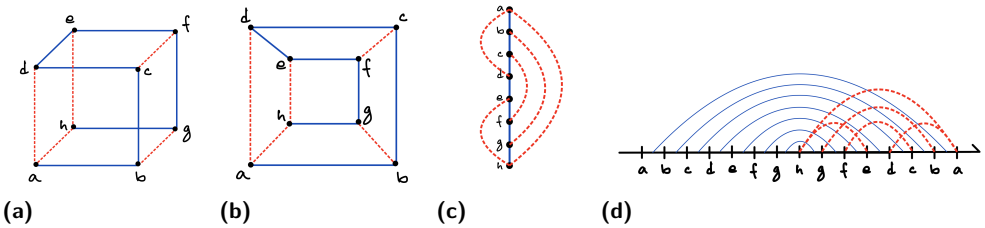

Figure 5 A cube design based on a path-like DFS tree. (a) DFS tree and co-tree of the cube mesh. (b) Corresponding Schlegel diagram. (c) DFS tree with back edges. (d) Arc diagram of the resulting design.

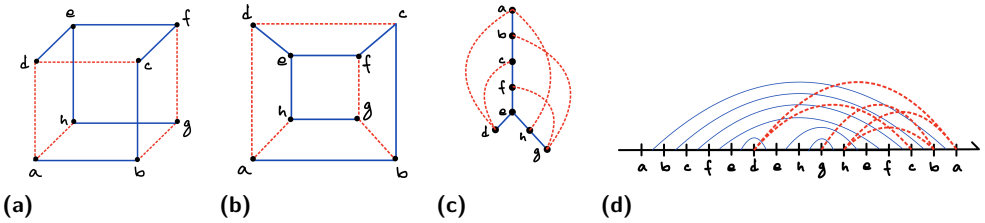

Figure 6 A cube design based on a branching DFS tree. (a) DFS tree and co-tree of the cube mesh. (b) Corresponding Schlegel diagram. (c) DFS tree with back edges. (d) Arc diagram of the resulting design.

tree, and $\delta : V \to [1..n]$ is a *pre-ordering* that labels the vertices in order of their first visits in the traversal process. Because of the depth-first manner of construction, any edge of G is either a part of the stem of T or connects a vertex and its ancestor, that is, some vertex on the unique path in T from the vertex to the root.

Note that the tree T can be embedded in the plane in such a way that the children of each vertex are ordered from left to right in increasing order of their δ values. A *pre-order traversal* of T in this embedding involves tracing its contour, starting from the root r, and writing down the label of each vertex at every visit. This sequence of labels is called a *DFS arrangement* of the vertices. For example, all the DFS trees of a tetrahedron are isomorphic (each is a simple 4-path), and the corresponding DFS arrangement is $1, 2, 3, 4, 3, 2, 1$ (see Figures 4(a) and (c)).

Observe that every edge of a graph G occurs exactly twice in any DFS arrangement of G: the first occurrence corresponds to traversing the edge forward when tracing the contour, and the second corresponds to traversing it backward later. Hence all DFS arrangements of a graph $G = (V, E)$ are of length $2|V| - 1$.

In the proposed use of a (rooted, ordered) DFS tree $T = (V, S, r, \delta)$ as a scaffold whose contour is traced by a single-stranded RNA w co-transcriptionally, each edge of T is to be assembled as an anti-parallel RNA helix, made of two (non-overlapping) complementary factors of w, while any other edge of G, which is a back edge as T is a DFS tree, is to be spanned by a kissing loop, which is made of two hairpins, sticking out of the both endpoints. A tree edge is spanned by one of the factors while being traversed forward, and remains single-stranded until it is wound around by the complementary factor into the helix. We say that a tree edge is *opened* when it is traversed forward, and *closed* once it has been traversed backward, that is, when it is completed as a helix. In contrast, we say that a back edge is *opened* once one of its hairpins has formed, while it is *closed* when both of them are "ready."

In order to be completed as a blueprint of a co-transcriptional folding pathway, the

DFS arrangement is to be augmented with edges as "arcs" from their time point of being opened to that of being closed. Tree edges can be thus added uniquely since every tree edge occurs exactly twice there. Now, it suffices to add back edges. Multiple occurrences of internal vertices (neither a root nor a leaf) provide us with freedom to choose, for each back edge (a, b), at which visits of a and b to have its two hairpins form. It is known to be kinetically unfavorable to close a cycle at a tree edge. This experimental obstacle motivates the "big-endian" principle. Recall that the vertices a and b are labelled with distinct integers $\delta(a)$ and $\delta(b)$, respectively, and whichever labelled smaller is an ancestor of the other. This principle says that, if $\delta(a) < \delta(b)$, then the assembly of this edge (by a kissing loop) should begin at b and end at a.

▶ **Lemma 1.** *The big-endian principle prevents any cycle of G from being closed at a tree edge.*

Proof. Any cycle of G involves a back edge as T is acyclic. Let E' be the set of edges in this cycle. Hence, the following inequality is well-defined:

$$\min_{(a,b)\in E'\setminus S}\{\min(\delta(a),\delta(b))\} \leq \min_{(u,v)\in E'\cap S}\{\min(\delta(u),\delta(v))\},$$

and certainly holds. Thus, this cycle is closed at a back edge that is incident to the vertex with the smallest value of δ among the vertices of the cycle. ◀

The BE (big-endian/back edge) principle does not yet fully eliminate the freedom in drawing an arc for (a, b) since b is visited more than once before the last visit at a, unless b is a leaf. In terms of the kissing loop crossing (KLX) number, discussed shortly, the arc should be drawn as short as possible, that is, from the last occurrence of b to the immediate occurrence of a (as a is an ancestor of b, the search returns to a after the last visit to b). However, this criterion of KLX optimisation does not pay any attention to possible adverse topological effect of focusing a lot of hairpin formations at one time point. Let us hence leave this freedom in the following formalisation of arc diagram. A *(BE-)arc diagram* of $G = (V, E)$ is a pair of a DFS arrangement $p_1, p_2, \ldots, p_m$ based on a DFS tree $T = (V, S, r, \delta)$ of G and a mapping $\alpha : E \setminus S \to [1..m] \times [1..m]$ with $m = 2|V| - 1$ such that for all back edge $e = (a, b) \in E \setminus S$ with $\delta(a) < \delta(b)$, if $\alpha(e) = (o, c)$, then $o < c$, $p_o = b$, and $p_c = a$. As $|S| = |V| - 1$, all arc diagrams of G are provided with $|E| - |V| + 1$ arcs for kissing loops.

An arc diagram of the tetrahedron is shown in Figure 4 and those of the cube without any branch and with a branch are shown respectively in Figures 5 and 6, where the arcs for kissing loops are coloured in red, while those for the tree stem are in blue, though they are not explicitly included in the formalisation.

4 Minimising kissing loop crosstalk

A set of kissing loop types should be as orthogonal as possible in the sense that mismatching hairpins barely hybridise. However, sets of kissing loops that have proven orthogonal enough in the laboratory are limited in size (see, e.g., [9]). The size of largest orthogonal KL sets available in reality serves as a standard for deciding whether a specific (rooted, ordered) DFS tree should be chosen or not. If the tree leaves more than this number of kissing loops open simultaneously at any point of folding, it should not be chosen. Recall that any DFS arrangement corresponds one-to-one with a rooted and ordered DFS tree. Given a BE-arc diagram $D = ((p_1, \ldots, p_m), \alpha)$ of a DFS tree $T = (V, S, r, \delta)$ of a graph $G = (V, E)$, the

KLX number of a segment (p_i, p_{i+1}) is the number of arcs that cross the vertical line drawn between p_i and p_{i+1}. It is defined formally as

$$\chi(p_i, p_{i+1}) = \big|\{e \in E \setminus S \mid \alpha(e) = (o, c), o \leq i, i+1 \leq c\}\big|. \tag{1}$$

The maximum of these values across all segments is the *KLX number of this diagram* D, that is, $\chi(D) = \max_{1 \leq i < m}\{\chi(p_i, p_{i+1})\}$. Finally, the *KLX number of the graph* G, denoted by $\chi(G)$, is the minimum among the KLX numbers of all the possible BE-arc diagrams of G.

The 1-to-1 correspondence between DFS arrangements and pairs of a graph and its rooted and preordered DFS tree justifies the introduction of the notation $\chi(G, T)$ as an alias of $\xi(G)$.

▶ **Lemma 2.** *Let $G = (V, E)$ and $T = (V, S, r, \delta)$ be its rooted DFS tree. Let T' be a (connected) subtree of T, and G' be the subgraph of G induced by the vertex set of T'. Then $\chi(G', T') \leq \chi(G, T)$.*

Proof. It is known that T' becomes a DFS tree of G' [12]. Indeed, it suffices to traverse T' according to the preorder δ. Note that T' may preorder the vertices differently and more favorably for the KLX number. ◀

This lemma can be used to prune the search tree for DFS trees with small KLX number, as outlined in Section 6.

As an algorithmic tool, it is useful to exclude some back edges from computing of the KLX number. For a subset of back edges $B \subseteq E \setminus S$, the KLX number of the segment (p_i, p_{i+1}) restricted to B, denoted by $\chi_B(p_i, p_{i+1})$, can be computed by replacing the occurrence of $E \setminus S$ in Eq. (1) with B. It is also convenient to define the KLX number of a tree edge $e \in S$ as the number of kissing loops that are opened but yet to be closed during the backward traversal across the edge; the following inequality justifies this definition.

▶ **Lemma 3.** *In the setting above, let (p_i, p_{i+1}) and (p_j, p_{j+1}) be the segments that correspond to the forward and backward traversals through an edge (u, v) of T, that is, $p_i = p_{j+1} = u$ and $p_{i+1} = p_j = v$. Then $\chi(p_i, p_{i+1}) \leq \chi(p_j, p_{j+1})$.*

Proof. In order for this inequality not to hold, there must be an arc (o, c) that crosses the segment (p_i, p_{i+1}) but not (p_j, p_{j+1}), that is, $o \leq i$ and $i + 1 \leq c \leq j$. Then $\delta(p_o) < \delta(p_c)$ would hold, but this contradicts the big-endian principle. ◀

▶ **Example 4** (KLX number of the cube)**.** See an arc diagram of the cube in Figure 5 (d); $\chi(h, g) = \chi(b, a) = 2$, $\chi(g, f) = \chi(c, b) = 3$, $\chi(f, e) = \chi(d, c) = 4$, and $\chi(e, d) = 3$, and therefore, the KLX number of this diagram is 4. Compare this with another diagram of the cube in Figure 6 (d), whose KLX number is 5. Consequently, the KLX number of the cube is at most 4.

Recall that, with one DFS tree fixed along with the preorder δ, any back edge (a, b) with $\delta(a) < \delta(b)$ should be opened at the last visit to b and then closed ASAP, that is, at the next visit to a. No other timing of opening/closing this edge that respects the big-endian principle improves in terms of KLX minimisation.

Given a rooted DFS tree $T = (V, S, r)$ without any preorder specified, the sibling order with the minimum KLX number can be computed bottom-up. Consider a branch v with its siblings $v_1, \ldots, v_d$, and suppose that they are visited in this order: v_1 first, v_2 next, and so on. Then any back edge between the subtree rooted at v_i and a vertex strictly above v increments by 1 the KLX number of all the segments corresponding to the edges

$(v, v_{i+1}), (v, v_{i+2}), \cdots, (v, v_d)$ or all the subtrees below them, although this contribution may not be very clear on the drawing of a DFS tree annotated with back edges, unless the tree is without any branch. Compare (c) with (d) in Figure 6; the back edges (d, a) and (d, c) even cross the segments that correspond to the whole traversal of the other subtree of e, which consists of the edges (e, h) and (h, g); this is as clear as day on the arc diagram. The subtrees below $v_1, \cdots, v_{i-1}$ are spared this increment because they have been fully explored before such an edge is opened. The back edges that cross over v increase the KLX numbers of the path from the root to v independently of the order in which the children of v are visited. These observations allow the KLX-minimum sibling orders for a given rooted but unordered tree to be computed in a *bottom-up* manner, starting from leaves, by comparing at each branch (the factorial number of) all permutations of its children.

Let v be a vertex with k children $v_1, v_2, \ldots, v_d$, T_v be the subtree of T below v, and T_i be the subtree of T below v_i. Let B be the set of back edges one of whose endpoints is in T_v, and let B_i be defined analogously with respect to T_i. Suppose that for all edges e in T_i with $1 \leq i \leq k$, the KLX numbers restricted to the back edges opened below v_i, that is, $\chi_{B_i}(e)$, have already been calculated. Let us compute the KLX number restricted less severely to the back edges opened below v. The back edges that come from inside T_i and go outside, that is, towards the path of T from the root to v can be categorised into those ending at v and those that go beyond; let us count them and denote the counts, respectively, by $\chi(T_i, v)$ and by $\chi(T_i, > v)$. Suppose that the children $v_1, \ldots, v_d$ of v are visited in this order. The KLX number of an edge e in T_i would be then incremented by $\sum_{k<i} \chi(T_k, > v)$, that is,

$$\chi_B(e) = \chi_{B_i}(e) + \sum_{k<i} \chi(T_k, > v). \tag{2}$$

That of an edge (v, v_i) would be set as

$$\chi_B((v, v_i)) = \left(\sum_{k<i} \chi(T_k, > v) \right) + \chi(T_i, v) + \chi(T_i, > v). \tag{3}$$

With the maximum among these numbers in Eqs. (2) and (3), this order competes with the others, and the children of v should be ordered according to the one that achieves the minimum; then the KLX number restricted to the back edges opened below v should be updated for all tree edges below v accordingly.

Ordering the children of even a single vertex in this way may require time factorial in $|V|$. For the class of 3 regular graphs, quadratic time suffices as a vertex can have at most two children.

5 The minimum kissing loop crossing and minimum tree depth problems

The *minimum KLX (kissing loop crossing) number problem* (MinKLX) asks, given an undirected, connected graph G and a positive integer k, if there exists a BE-arc diagram of G whose KLX number is at most k.

▶ **Theorem 5.** *The* MinKLX *problem is NP-hard.*

Proof. The proof is based on the proof by Gavril [4] for the NP-hardness of computing the cutwidth of a graph $G = (V, E)$, which asks, given also an integer k, to arrange the vertices of G along a horisontal line in such a way that, for any vertical line drawn between adjacent vertices, diving V into those to its left and those to its right,

The reduction is from Max Cut, which asks to split the vertex set V of a given weighted graph $G = (V, E)$ into two subsets $V' \subseteq V$ and $V \setminus V'$ so as to maximise the sum of the weights of edges that connect these subsets in G. Given a pair (G, w) of a n-vertices graph $G = (V, E)$ and a positive integer w, let us convert this instance of max cut into an instance of KLX computation problem $(\overline{G}, k)$ as follows. With a "large enough" r, let $U = \{u_1, u_2, \ldots, u_r\}$ be a set of auxiliary "universal" vertices. Let $\overline{G} = (V \cup U, \overline{E})$ with $\overline{E} = ((V \cup U) \times (V \cup U)) \setminus E$. Note that, for any $x \geq 1$, all DFS trees of the complete graph K_x are equivalent, they are indeed a path (no branch), and their KLX number $f(x) = \lceil x/2 \rceil \times \lfloor x/2 \rfloor$ can be computed in polynomial time. Let $k = f(n + r) - w$. Now we are ready to show that G has a cut (A, B) with at least w edges between A and $B = V \setminus A$ if and only if the KLX number of $\overline{G}$ is at most k.

Firstly, suppose $\overline{G}$ has such a cut, and let $A = \{a_1, a_2, \ldots, a_m\}$ and $B = \{b_1, b_2, \ldots, b_{n-m}\}$. The linear arrangement of $V \cup U$

$$a_1, u_1, a_2, u_2, \cdots, u_{m-1}, a_m, u_m, u_{m+1} \cdots u_{r-(n-m-1)}, b_1, u_{r-(n-m-1)+1}, \cdots, u_r, b_{n-m}$$

amounts to a DFS tree of $\overline{G}$ thanks to a universal "glueing" vertex between a_i and a_{i+1} (or b_j and b_{j+1}), which are not necessarily connected in $\overline{G}$ (indeed, by definition, they are not connected in $\overline{G}$ iff they are connected in G). With large enough r, a tree edge that is crossed by the largest number of back edges is located in the interval $u_m, \ldots, u_{r-(n-m-1)}$, and this number is at most k. Thus, the KLX number of $\overline{G}$ is at most k.

For the opposite implication, suppose that $\overline{G}$ has a DFS tree T whose KLX number is at most k. This tree must be "almost" a path in the sense that below a branch, if any, no universal vertex can appear in order not to introduce any cross edge between subtrees below the branch. Therefore, the path from the root of T to its first branch, if any, is of length at least r, and since r is large enough, an edge e that determines the KLX number of this DFS tree is somewhere along this path. This path can be extended into a DFS path P of the complete graph K_{n+r}. The edge which is crossed by $f(n + r)$ back edges is on this path. Among these back edges, at most k of them belong to $\overline{E}$, and the others are edges of the original graph; let E_1 be the subset of E that consist of these edges. Then, $f(n+r) \leq k + |E_1|$, which implies $|E_1| \geq f(n+r) - k = w$. Let S be the set of vertices in V that occur above this edge. Then $(S, V \setminus S)$ is a cut which is crossed by at least w edges. ◀

Another relevant problem is that of determining the depth of the shallowest DFS tree for a given graph, because finding a shallower DFS tree decreases the number of helical domains kept open in parallel, and may result in a sequence with fewer helical domain types. This quantity is known as the *tree-depth* of the graph, a measure of how far the graph is from being a star. Computing tree-depth (the MinTD problem) is NP-hard even for the class of triangulated graphs [2].

6 Solving the MinKLX and MinTD problems by enumeration

The NP-hardness of the MinKLX and MinTD problems may not leave us any algorithmic option but somehow enumerating DFS trees of a given graph $G = (V, E)$ to find a good co-transcriptional folding pathway, although the latter theorem is too weak to exclude the possibility that the KLX and TD numbers could be actually computable in a polynomial time for certain classes of graphs of practical significance such as that of 3-regular graphs, in which all the vertices have exactly 3 neighbours. As any graph can be approximated by a 3-regular one by replacing each vertex by a network of vertices of degree 3, it is highly prioritised to figure out whether these problems remain hard for this class. The related

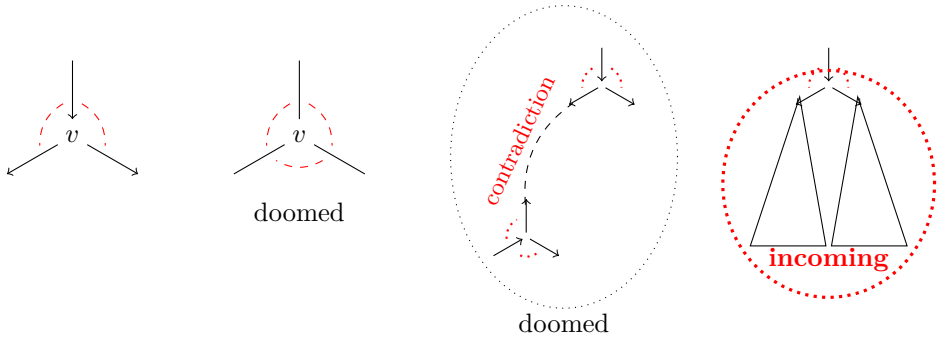

Figure 7 Intrinsic orientation of a branch, an internally contradicting block, and global prohibition of root by a single branch. Dotted red lines indicate a bypass around the center of a branch, which is necessary for a graph to be biconnected.

cutwidth problem, described in the proof of Theorem 5, is NP-hard for the class of planar graphs with maximum degree 3 [15] (the bipartite graph $K_{3,3}$ is an example of 3-regular, non-planar graph).

The graph $G = (V, E)$ and its free spanning tree $T = (V, S)$ can be uniquely decomposed into a set of biconnected components, or *blocks*, $G_i = (V_i, E_i)$, which cannot be disconnected by removing one vertex, along with a spanning tree $T_i = (V_i, S_i)$ induced by restricting T onto V_i, and these blocks are further organised into a so-called *block-cut tree* via edges between blocks that share (exactly one) vertex in common, which is an articulation point of G (removing it disconnects G); any articulation point of G thus serves as an interface between two or more blocks. The resulting blocks can be categorised into the following three classes:

- **Branching.** involving a branch of T, which consists of a vertex v and three edges in S that are incident to v;
- **Spinal.** involving at least three vertices but not branching;
- **Edge.** consisting of one edge.

A spinal block has at most two interfaces, while a branching one can have more.

As no ancillary edge (of G but not in T) crosses an articulation point or even an edge block, the KLX number of G according to T is the maximum of those of the branching and spinal blocks. The computation actually does not require T to be rooted, but suffices for these blocks to know at or beyond which interfaces of them is the global root of T. It is well known that a free DFS tree of a biconnected graph cannot be rooted anywhere but its leaf. This implies that in our problem setting, for T_i to be a DFS tree of the block G_i, it must be rooted at one of its interfaces with adjacent blocks or at its leaf. For this reason, at most one block that is either spinal or branching can accommodate the global root inside. The KLX number of a spinal block does not change even if the block is "flipped upside down," that is, regardless of beyond which of its possible two interfaces the global root is. In contrast, branching blocks deny free rotation by restricting where the global root can be due to intrinsic orientation of each of their branches, as we see from now on.

Once rooted, all edges of T become fully oriented; let us assume that orientations are from the root (top) towards the leaves (bottom). If this orientation makes T into a DFS tree, then the induced orientation by V_i makes T_i also into a DFS tree of G_i. This is, in fact, the only rooting that endows T_i with the DFS property due to the following critical observation by Korach and Ostfeld [12]: given a biconnected graph and its free spanning tree, if the tree has a branch, there exists at most one vertex where the tree can be rooted in such a way

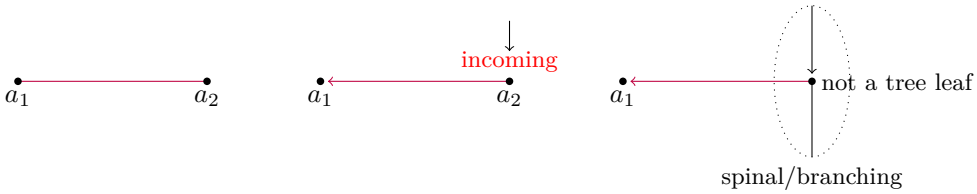

Figure 8 The three admissible edge additions between two trees: (left) between two articulation points that are not incoming; (middle) between two articulation points exactly one of which is incoming; (right) between a non-incoming articulation point and an internal vertex of a spinal or branching block that is not a tree leaf. In the latter two cases, the added edge is oriented towards the non-incoming articulation point, a_1 here, and makes the whole tree where the point is, that is, the left tree here, incoming, that is, prohibited from being rooted.

that the resulting rooted spanning tree is a DFS tree, and furthermore, the vertex must be a leaf of the spanning tree; without such a leaf, the tree cannot be obtained by any DFS regardless of where we being the search. Let us reproduce the essence of this result. Assume in contradiction that there are two such vertices a and b. If rooted at a, the branch has at least 2 subtrees T_1 and T_2, neither of which contains a; without loss of generality, we assume that T_1 does not contain b. Since the tree becomes a DFS tree by being rooted at a, and v is not an articulation point, there must be a backward edge from T_1 to some vertex above v along the unique tree path from v to the root a, but this would become a cross edge once the tree was rooted at b, a contradiction.

Every branch inside a branching block is intrinsically oriented; among the edges incident to it, one is incoming and all the others are outgoing, as illustrated in Figure 7, globally prohibiting any vertex beyond these outgoing edges from becoming the root by labeling them "incoming." At this point, a branch already proves itself ineligible to be part of a DFS tree if any two of its three subtrees come and go by bypass, indicated by red dashed lines. Let us emphasise that the branch cannot have all the edges incident to it be outgoing and become the root because T_i must be rooted at its leaf to become a DFS tree of G_i. These local orientations globally restrict where T can be rooted, unless they contradict each other, for example, by yielding a vertex with more than one incoming edge. In other words, any rooting of T that does not respect the intrinsic orientation of a single branch cannot be a DFS tree of G. This observation and the downward closedness of the KLX number along a specific spanning tree (Lemma 2) enable us to enumerate the spanning trees of G by recursively deciding whether an edge of G should be part of the spanning tree T under construction or not and, if chosen, added to a tree or between trees in a spanning forest that is part of T in an admissible manner (see Figure 8). An edge can be added to a tree unless it results in a cycle. In case of bridging two trees T_1 and T_2 by a (tree) edge, all the other edges between them are ruled out as a candidate of tree edges and introduced rather as an ancillary edge unless they result in a bad configuration; for example, if one of these ancillary edges is between u_1 and u_2, another is between v_1 and v_2, and all these four vertices are pairwise distinct, then the tree must rooted globally at u_1 or u_2 in order for the edge not to become a cross edge, and v_1 or v_2 claims the ownership of the global root analogously, but these two claims are obviously not compatible. Bridging two trees by an edge may bundle some of the blocks in T_1 and in T_2 into one. The resulting block can be computed efficiently [17] but its KLX number should also be computable more efficiently from those of the bundled blocks and from the cost due to the newly added ancillary edges than being computed from scratch.

The performance of such enumerative algorithms relies heavily on how efficiently they

Algorithm 2 Enumerative KLX minimisation for a graph $G = (V, E)$

Let $n = |V|$, $m = |E|$, and edges be indexed as $E = \{e_1, e_2, \ldots, e_m\}$

function KLXT(T) ▷ Compute KLX number of tree T
    return the maximum of the KLX numbers of the blocks of tree T
end function

function KLX(F, k)
▷ Compute min KLX number over all completions of forest F with edges in $\{e_k, \ldots, e_m\}$
    if F comprises a single tree with $n - 1$ edges **then**
        $klx_{\text{tree}} \leftarrow \text{KLXT}(F)$
        $klx_{\text{min}} \leftarrow \min\{klx_{\text{min}}, klx_{\text{tree}}\}$
        return klx_{tree}
    end if
    if edge e_k does not create a cycle and is admissible in F **then** ▷ See Figure 8
        $F' \leftarrow F \cup \{e_k\}$ ▷ e_k included as a tree edge
        let T be the tree that contains edge e_k in forest F'
        if $\text{KLXT}(T) \geq klx_{\text{min}}$ **then** ▷ Prune if cost of $T \geq klx_{\text{min}}$
            $klx_1 \leftarrow m$
        else
            $klx_1 \leftarrow \text{KLX}(F', k + 1)$
        end if
    end if
    $klx_0 \leftarrow \text{KLX}(F, k + 1)$ ▷ e_k not included as a tree edge
    return $\min\{klx_1, klx_0\}$
end function

$klx_{\text{min}} \leftarrow m$
return $\text{KLX}(\emptyset, 1)$ ▷ Start with an empty forest

prune unproductive branches. Both the KLX number of a forest and bad configurations should be utilised for pruning. The algorithms record the KLX-best DFS tree found so far along with its KLX number, and once a forest under construction exceeds the threshold, it should be rejected immediately. A forest with a bad local configuration should be rejected as well. The latter pruning requires a proper data-structure for early and effective detection of bad configurations. The number of available orthogonal sets of helical domains also spares the algorithms from redundant explorations as an upper bound on the tree-depth.

7 Examples

The cotranscription-friendly DFS-tree based design method presented in Section 3 is implemented and available for use in the online design tool *DNAforge* (`https://dnaforge.org`), together with an option for minimising the KLX cost of the design with a preliminary version of the enumeration method presented in Section 6.[2]

[2] Design method ST-RNA, additional parameters "co-transcriptional route" and "minimise the number of kissing loop sequences"

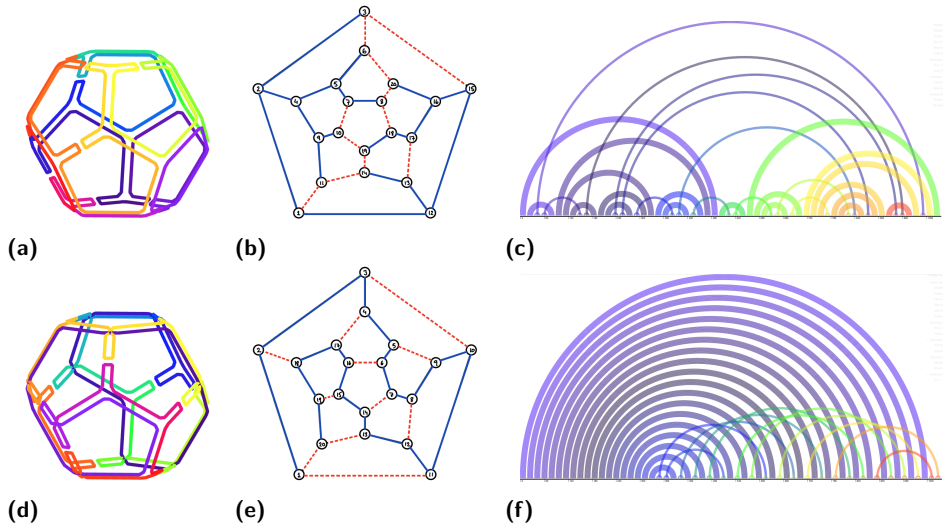

Figure 9 Upper row: a dodecahedron design based on a randomly chosen spanning tree. (a) Strand routing on the 3D wireframe. (b) Spanning tree (solid blue) and back edges (dashed red) on the Schlegel diagram. (c) Domain-level arc diagram. (Long helical domain pairings thick, short kissing-loop domain pairings thin.) Lower row (d)-(f) A dodecahedron design based on a KLX-optimised DFS spanning tree (KLX = 6).

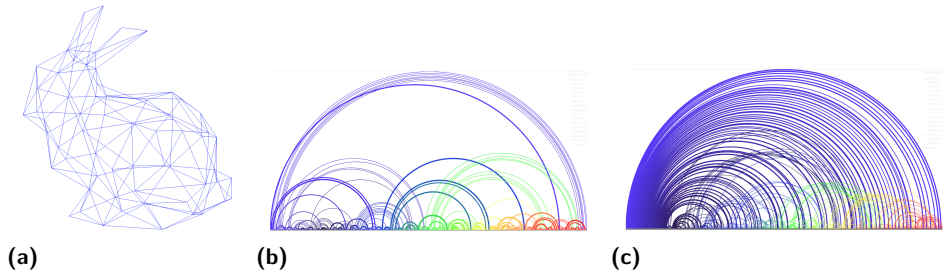

Figure 10 Designs for a 3D mesh model of a bunny. (a) Wireframe model. (b) Arc diagram of random spanning tree routing. (c) Arc diagram of KLX-optimised DFS spanning tree routing (KLX = 33).

Figures 9 and 10 illustrate some outcomes from the tool. Figure 9 shows designs of wireframe dodecahedra based on a randomly chosen spanning tree (upper row) and a DFS spanning tree (lower row). The DFS-tree based design has also been KLX-optimised, resulting in a reduction from a KLX number of 9 in the initial DFS tree to 6 in the optimal one. (The spanning tree diagrams in Figures 9(b) and (e)) have been manually reconstructed from the tool-generated diagrams in Figures 9(c) and (f).) Figure 10 displays random-tree and DFS-tree designs for a 66-vertex, 192-edge wireframe model of a bunny. Also here the DFS-tree based design has been KLX-optimised, resulting in a KLX number reduction from 60 in the initial DFS tree to 33 in the optimal one. Table 1 summarises the KLX number reductions for some basic mesh models.

8 Conclusions and further work

We have presented models and algorithms for addressing two tasks in secondary structure design for cotranscriptionally folding DNA origami wireframe nanostructures: avoiding the

Model	Vertices	Edges	Initial KLX	Min KLX
Tetrahedron	4	6	3	3
Cube	8	12	4	4
Octahedron	6	12	6	5
Dodecahedron	20	30	9	6
Icosahedron	12	30	12	10
Bunny	66	192	60	33

Table 1 Effect of KLX minimisation on some 3D mesh models.

topological folding obstacle of polymerase trapping and minimising the number of distinct kissing loop designs (the KLX number). The key tools in this work have been the algorithmic method of depth-first search in graphs and the ensuing DFS spanning trees. Our branch-and-bound approach to the KLX minimisation problem can also be used for any other effectively computable objective function on DFS trees, such as the DFS tree depth of a given graph (the TD number).

Relevant directions for further work include for instance the following:

1. Nucleotide-level sequence design for DNA origami wireframes in the cotranscriptional setting.
2. Efficient combinatorial algorithms for minimising the KLX and TD numbers in some interesting classes of graphs, such 3-regular or polyhedral graphs, or proving the problems NP-hard in these classes.
3. Efficient fixed-parameter or approximation algorithms for minimising the KLX and TD numbers in some relevant classes of graphs.
4. Improved enumeration of DFS trees in polyhedral graphs.

References

1 Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*. MIT Press, Cambridge MA, USA, 4th edition, 2022.

2 Dariusz Dereniowski and Adam Nadolski. Vertex rankings of chordal graphs and weighted trees. *Information Processing Letters*, 98(3):96–100, 2006.

3 Antti Elonen, Ashwin Karthick Natarajan, Ibuki Kawamata, Lukas Oesinghaus, Abdulmelik Mohammed, Jani Seitsonen, Yuki Suzuki, Friedrich C. Simmel, Anton Kuzyk, and Pekka Orponen. Algorithmic design of 3D wireframe RNA polyhedra. *ACS Nano*, 16:16608–18816, 2022. doi:10.1021/acsnano.2c06035.

4 Fănică Gavril. Some NP-complete problems on graphs. In *Proceedings of the 1977 Conference on Information Sciences and Systems*, pages 91–95, 1977. URL: `https://scispace.com/papers/some-np-complete-problems-on-graphs-4l02n07tlh`.

5 Cody Geary, Guido Grossi, Ewan K. S. McRae, Paul W. K. Rothemund, and Ebbe S. Andersen. RNA origami design tools enable cotranscriptional folding of kilobase-sized nanoscaffolds. *Nature Chemistry*, 13(6):549–558, 2021. doi:10.1038/s41557-021-00679-1.

6 Cody Geary, Pierre-Étienne Meunier, Nicolas Schabanel, and Shinnosuke Seki. Programming biomolecules that fold greedily during transcription. In *41st International Symposium on Mathematical Foundations of Computer Science*, pages 43:1–43:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2016. doi:10.4230/LIPIcs.MFCS.2016.43.

7 Cody Geary, Paul W. K. Rothemund, and Ebbe S. Andersen. A single-stranded architecture for cotranscriptional folding of RNA nanostructures. *Science*, 345(6198):799, 2014. doi:10.1126/science.1253920.

8 Cody W. Geary and Ebbe Sloth Andersen. Design principles for single-stranded RNA origami structures. In *DNA Computing and Molecular Programming: 20th International Conference*, pages 1–19. Springer, 2014. doi:10.1007/978-3-319-11295-4_1.

9 Wade W. Grabow, Paul Zakrevsky, Kirill A. Afonin, Arkadiusz Chworos, Bruce A. Shapiro, and Luc Jaeger. Self-assembling RNA nanorings based on RNAI/II inverse kissing complexes. *Nano Letters*, 11(2):878–887, 2011. doi: 10.1021/nl104271s.

10 Peixuan Guo. The emerging field of RNA nanotechnology. *Nature Nanotechnology*, 5(12):833–842, December 2010. doi: 10.1038/nnano.2010.231.

11 Daniel Jasinski, Farzin Haque, Daniel W. Binzel, and Peixuan Guo. Advancement of the emerging field of RNA nanotechnology. *ACS Nano*, 11(2):1142–1164, 2017. doi: 0.1021/acsnano.6b05737.

12 Ephraim Korach and Zvi Ostfeld. DFS tree construction: Algorithms and characterizations. In *Graph-Theoretic Concepts in Computer Science*, pages 87–106. Springer Berlin Heidelberg, 1989. doi:10.1007/3-540-50728-0_37.

13 Di Liu, Cody W. Geary, Gang Chen, Yaming Shao, Mo Li, Chengde Mao, Ebbe S. Andersen, Joseph A. Piccirilli, Paul W. K. Rothemund, and Yossi Weizmann. Branched kissing loops for the construction of diverse RNA homooligomeric nanostructures. *Nature Chemistry*, 12(3):249–259, 2020. doi:10.1038/s41557-019-0406-7.

14 Abdulmelik Mohammed, Pekka Orponen, and Sachith Pai. Algorithmic design of cotranscriptionally folding 2D RNA origami structures. In *Unconventional Computation and Natural Computation: 17th International Conference*, pages 159–172. Springer, 2018. doi: 10.1007/978-3-319-92435-9_12.

15 Burkhard Monien and Ivan Hal Sudborough. Min cut is NP-complete for edge weighted trees. *Theoretical Computer Science*, 58(1):209–229, 1988. doi:10.1016/0304-3975(88)90028-X.

16 Erik Poppleton, Niklas Urbanek, Taniya Chakraborty, Alessandra Griffo, Luca Monari, and Kerstin Göpfrich. RNA origami: design, simulation and application. *RNA Biology*, 20(1):510–524, 2023. doi:10.1080/15476286.2023.2237719.

17 Jeffery Westbrook and Robert E. Tarjan. Maintaining bridge-connected and biconnected components on-line. *Algorithmica*, 7(1):433–464, 1992. doi: 10.1007/BF01758773.