


# 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,  
Tokyo, 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. Key concepts in both cases are depth-first search in graphs and the ensuing DFS spanning trees. 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

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

**Digital Object Identifier** 10.4230/LIPIcs.DNA.31.6

**Funding** This work was initiated during a visit by P.O. to the University of Electro-Communications, supported by grant “Design Tools for DNA Nanotechnology” from the Finnish Cultural Foundation, and significantly advanced during a visit by S.S. to Aalto University, supported by a Visiting Researcher grant from the Aalto Science Institute.

**Acknowledgements** We thank Mr. Robin Runne for essential contributions to the implementation of our DFS tree enumeration method.

## 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



© Pekka Orponen, Shinnosuke Seki, and Antti Elonen;  
licensed under Creative Commons License CC-BY 4.0

31st International Conference on DNA Computing and Molecular Programming (DNA 31).

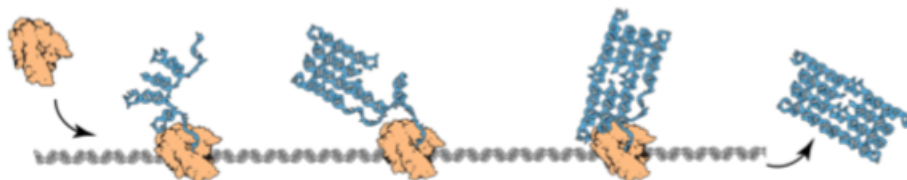
Editors: Josie Schaeffer and Fei Zhang; Article No. 6; pp. 6:1–6:18

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

variety of pseudoknots [11, 12]. 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 [8]. 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 [14, 6]. (For an overview, see [17].)



**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 [7].)

One emphasis in the work of Geary et al. [8, 6] 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.

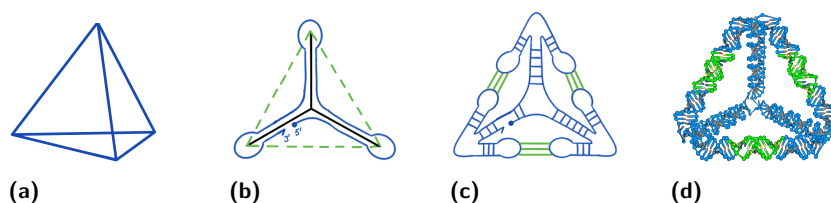
In the following, Section 2 presents a spanning-tree based framework for self-assembly of wireframe structures by co-transcriptional folding, and introduces the topological folding obstacle of polymerase trapping. Section 3 then demonstrates how this obstacle can always be avoided by using a depth-first-search (DFS) based design scheme. Next, Section 4 introduces the notion of the KLX number of a DFS tree, which 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 efficiently reusing KL designs, although, as proved in Section 5, the KLX minimisation problem is in the general case NP-hard. Since an efficient minimisation algorithm is thus unlikely, Section 6 provides a branch-and-bound type enumeration approach to 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. [15]). The general spanning-tree based 3D wireframe design scheme is outlined in Figure 2.

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  $G$ ,<sup>1</sup> 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

<sup>1</sup> A spanning tree of a graph  $G$  is an acyclic subgraph that connects all the vertices of  $G$ .

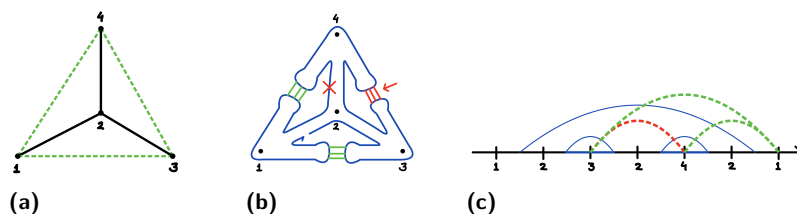


■ **Figure 2** A spanning-tree based design scheme for 3D RNA wireframe origami. (a) Targeted wireframe model. (b) A spanning tree and strand routing of the wireframe graph. (c) Routing-based stem and kissing-loop pairings. (d) Nucleotide-level model. (Adapted with permission from [3].)

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  $180^\circ$  kissing-loop design motif introduced in [8], thus constituting the co-tree edges. Figure 2(d) presents a nucleotide-level model of the eventual nanostructure.

One potentially significant topological obstacle to cotranscriptional folding in this framework is the phenomenon of *polymerase trapping*, first identified by Geary and Andersen in [9] and also addressed in the recent 2D RNA origami design tool ROAD by Geary et al. [6, pp. 551–552, SI p. 102 ff]. Our mathematical model of this phenomenon is basically the same as introduced and analysed by Mohammed et al. in [15], but adapted to the present setting of 3D wireframe origami designs.

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 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 green lines.



■ **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.

Figure 3(b) depicts again the corresponding twice-around-the-tree strand routing (blue) and the complementary kissing-loop pairings (green and 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.

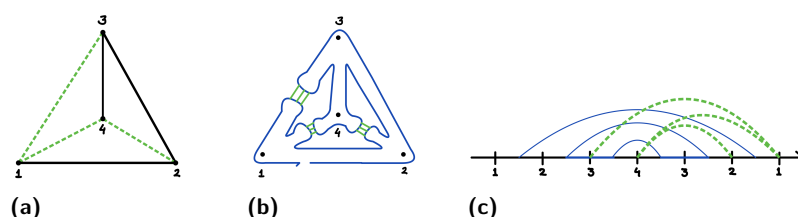
The schematic in Figure 3(c) presents the design as a domain-level arc diagram, where the strand is laid out along a line in the  $5'$  to  $3'$  direction, the vertex visits are marked by the corresponding indices, the domain-to-domain helical pairings are indicated by solid blue arcs, and the kissing-loop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

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 red 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 (drawn in red and 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.<sup>2</sup>)

Schematically, one can see that the risk of this kind of “polymerase trapping” obstacle emerges when a kissing-loop pair, 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 “dashed arc” that has been initiated before a “solid arc” terminates inside that solid arc.



■ **Figure 4** A tetrahedron design based on a 4-vertex 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.

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 have been initiated after the kissing loop was opened.

Such complete absence of polymerase traps seems like a very particular property, and one may wonder for which kinds of wireframe models this situation 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].

<sup>2</sup> Furthermore, moving from our strand-centric to a polymerase-centric view, the polymerase of course does not stop transcribing even if the folding is temporary blocked. Eventually the polymerase detaches from the fully transcribed strand and leaves it to fold the unfolded 3' tail the best it can. From this perspective, the polymerase trapping phenomenon is more of a kinetic than a topological obstacle.



### 3 Cotranscription-friendly secondary structure design

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

---

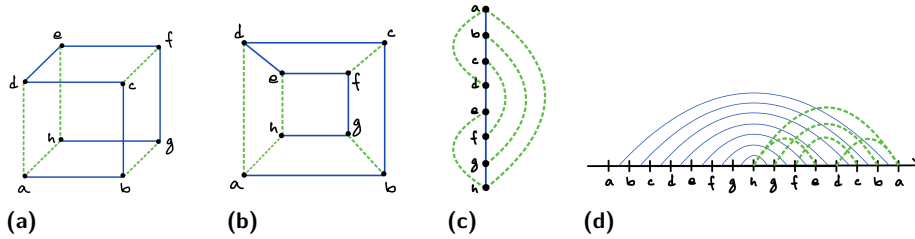
```

1: Initially all vertices  $v \in V$  and edges  $e \in E$  are set to be unmarked.
2:
3: function DFS( $G, r$ )
4:   mark vertex  $r$  as visited
5:   for each edge  $e = \{r, v\}$  incident to  $r$  do
6:     if vertex  $v$  is not marked as visited then
7:       mark  $e$  as a tree edge
8:       perform search DFS( $G, v$ )
9:     else
10:      mark  $e$  as a back edge, unless it is already marked (= edge to parent)
11:    end if
12:  end for
13: end function

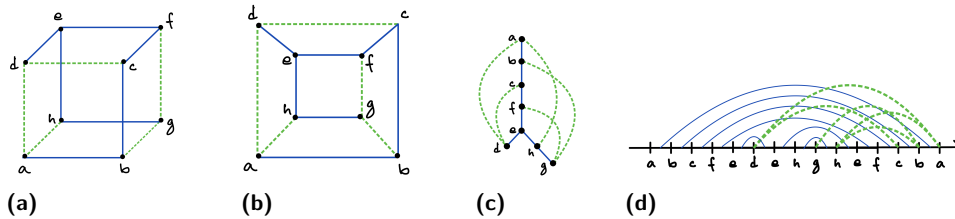
```

---

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.



■ **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.

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, whereas 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 [13]; in other words, there are no “cross edges” connecting two different branches of the directed tree, as exemplified in Figures 5 and 6 (c), and also in Figure 7.<sup>3</sup>

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 edges,  $r \in V$  is the chosen root which determines the orientation of the tree, and  $\delta : V \rightarrow [1..n]$  is a *pre-ordering* that labels the vertices in order of their first visits in the traversal. As discussed above, any edge of  $G$  is either part of the stem  $S$  or connects a vertex and its ancestor along the unique path in  $T$  from the vertex to the root. We often consider  $T$  as embedded in the plane so that the children of each vertex are ordered from left to right in increasing order of their  $\delta$  values. Thus, for example, vertex  $d$  in Figure 6(c) is presumed to have been visited earlier than vertex  $h$  in the traversal that created the tree.

A plane embedded DFS tree  $T = (V, S, r, \delta)$  provides an initial blueprint for designing a cotranscriptionally folding RNA wireframe nanostructure. In the first stage of the design process, the contour of the tree  $T$  is traced by an RNA strand  $w$  so that each edge of  $T$  becomes assembled as an RNA helix, hybridised from two complementary antiparallel domains of  $w$ . In the second stage the co-tree edges of  $G$ , which are now back edges of the DFS tree  $T$ , are constituted as kissing loops made of two half-edge hairpins that extend to meet from the end-vertices of the respective edges.

Let us call the sequence of vertex labels encountered during a walk around the contour of any labelled, plane embedded tree  $T$  a *contour trace* or *linear arrangement* of  $T$ .<sup>4</sup> In the special case that  $T$  is a DFS tree we refer to this sequence as a *DFA arrangement* based on  $T$ . By construction, any contour trace of a tree  $T$  accommodates every edge  $e$  of  $T$  exactly twice: firstly, when the traversal crosses  $e$  in a forward direction, and secondly when the traversal crosses  $e$  in the opposite direction. Since any DFS tree (or more generally any spanning tree) of a graph  $G$  with  $n$  vertices has  $n - 1$  edges, any DFS arrangement in  $G$  contains  $2n - 1$  vertex labels. For example, every DFS tree of a tetrahedron is a simple 4-path in its Schlegel diagram, and the corresponding DFS arrangement has the pattern 1, 2, 3, 4, 3, 2, 1 (see Figures 4(a) and (c)).

To complete our blueprint for designing a cotranscriptionally folding RNA nanostructure, the DFS arrangement needs to be augmented into a (domain-level) *arc diagram* by adding information that indicates when, for each pair of hybridising domains  $(\alpha, \alpha^*)$ , the forward domain  $\alpha$  is transcribed and when its complementary reverse domain  $\alpha^*$ . For each such pair, these time points are connected by an edge (“arc”), with the first time point considered as the “opening” and the second the “closing” time for this pair. Since in our design scheme, DFS tree edges correspond to helical domains, we correspondingly say that a tree edge is *opened* when it is crossed by the contour traversal in the forward direction, and *closed* when it is crossed in the reverse direction. Note that the chosen DFS completely determines the scheduling of the opening and closing pattern of the helical domains.

Let us then consider scheduling the opening and closing times of the back edges, i.e. kissing loops in the eventual RNA nanostructure. 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  $b$  each of the two constituent hairpins of it is to be formed. For each

<sup>3</sup> The simple reason for the absence of cross edges is that if for an edge  $e = \{u, v\}$ , vertex  $v$  is still unmarked when the traversal first considers  $e$  at vertex  $u$  (or vice versa), then  $e$  becomes a tree edge.

<sup>4</sup> Also known as a “twice-around-the-tree walk” and *full walk* in [1, p. 1112].

back edge  $(a, b)$ , an occurrence of  $a$  and that of  $b$  can be chosen and connected by an arc; then we say that this edge is *opened* at the time point which corresponds to the left end of the arc, and *closed* at the right end.

The freedom in timing opening and closing of back edges brings multiple arc diagrams based on the same DFS arrangement. Which should we choose then? Given an arc diagram of  $G$ , we say that a cycle in  $G$  is *closed at an edge  $e$*  if  $e$  is in the cycle, and in the arc diagram, all the other edges involved in the cycle have already been closed before  $e$  is closed. It is known to be kinetically unfavourable to close a cycle at a tree edge. This experimental obstacle motivates our “phloem principle” for ordering back edge connections.<sup>5</sup> 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 phloem principle prevents any cycle of  $G$  from being closed at a tree edge.*

**Proof.** Any cycle of  $G$  involves a back edge as  $T = (V, S, r, \delta)$  is acyclic. Let  $E'$  be the set of edges in this cycle; then  $E' \cap S$  and  $E' \setminus S$  are the set of tree edges in the cycle and the set of back edges in the cycle, respectively. The absence of cross edges implies that subtrees of a vertex are connected only via the vertex even in  $G$ . Hence, the vertex with the smallest  $\delta$ -value in a cycle must be incident to a back edge in the cycle; in other words, the following inequality must hold:

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

Thus, this cycle is closed at the back edge. ◀

This lemma ensures that arc diagrams serve as a blueprint of the polymerase-trap-free co-transcriptional folding pathway as long as they obey the phloem principle. Now we count out any arc diagram that violates the principle. Thus, from now on, an arc diagram of  $G = (V, E)$  is a pair of a DFS arrangement  $p_1, p_2, \dots, p_m$  based on a DFS tree  $T = (V, S, r, \delta)$  of  $G$  and a mapping  $\alpha : E \setminus S \rightarrow [1..m] \times [1..m]$  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  $p_c = a$ . (Arcs for tree edges do not appear anywhere in this definition, but they are uniquely identified by the DFS arrangement.) 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 (d) and those of the cube without any branch and with a branch are shown respectively in Figures 5 (d) and 6 (d), where the arcs for kissing loops are coloured in green, while those for the tree stem are in blue. All of them follow the phloem principle.

The phloem principle does not fully eliminate the freedom in drawing an arc for  $(a, b)$  with  $\delta(a) < \delta(b)$  since  $b$  is visited more than once before the last visit at  $a$ , unless  $b$  is a leaf. For the sake of the kissing loop crossing (KLX) number, a measure of how good an arc diagram is that we shall discuss next, 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 the possible adverse topological effect of focusing a lot of hairpin formations at one time point. Therefore, we have left this freedom in the above definition of arc diagram.

<sup>5</sup> Phloem is a pathway for transporting products of photosynthesis from leaves to the rest of a plant.

#### 4 Minimising kissing loop crosstalk

A set of kissing loop types should be as orthogonal as possible to minimise the risk of *crosstalk*, that is the risk of mismatching hairpins hybridising. However, sets of kissing loops that have proven orthogonal enough in the laboratory are limited in size (see, e.g., [10]). 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 design leaves more than this number of kissing loops open simultaneously at any point of folding, it increases the risk of crosstalk.

Recall that any DFS arrangement corresponds one-to-one with a rooted and ordered DFS tree. Given an arc diagram  $D = ((p_1, \dots, p_m), \alpha)$  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

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

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

► **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  $\kappa(G', T') \leq \kappa(G, T)$ .*

**Proof.** It is known that  $T'$  becomes a DFS tree of  $G'$  [13]. Indeed, it suffices to traverse  $T'$  according to the preorder  $\delta$ . 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  $\kappa_B(p_i, p_{i+1})$ , 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  $\kappa(p_i, p_{i+1}) \leq \kappa(p_j, p_{j+1})$ .*

**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 phloem principle. ◀

► **Example 4 (KLX number of the cube).** See an arc diagram of the cube in Figure 5 (d);  $\kappa(h, g) = \kappa(b, a) = 2$ ,  $\kappa(g, f) = \kappa(c, b) = 3$ ,  $\kappa(f, e) = \kappa(d, c) = 4$ , and  $\kappa(e, d) = 3$ , 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  $\delta$ , 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 phloem 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  $v_1, \dots, v_d$ , 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  $(v, v_{i+1}), (v, v_{i+2}), \dots, (v, v_d)$  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  $v_1, \dots, v_{i-1}$  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 domainial number of) all permutations of its children.

Let  $v$  be a vertex with  $k$  children  $v_1, v_2, \dots, v_d$ ,  $T_v$  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,  $\kappa_{B_i}(e)$ , 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  $\kappa(T_i, v)$  and by  $\kappa(T_i, > v)$ . Suppose that the children  $v_1, \dots, v_d$  of  $v$  are visited in this order. The KLX number of an edge  $e$  in  $T_i$  would be then incremented by  $\sum_{k < i} \kappa(T_k, > v)$ , that is,

$$\kappa_B(e) = \kappa_{B_i}(e) + \sum_{k < i} \kappa(T_k, > v). \quad (2)$$

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

$$\kappa_B((v, v_i)) = \left( \sum_{k < i} \kappa(T_k, > v) \right) + \kappa(T_i, v) + \kappa(T_i, > v). \quad (3)$$

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 domainial time 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 an arc diagram of  $G$  whose KLX number is at most  $k$ . This problem is computationally hard as stated in the next theorem; its proof can be found in the technical appendix A.

► **Theorem 5.** *The MINKLX problem is NP-hard.*

Another relevant problem is that of determining the depth of the shallowest DFS tree for a given graph  $G$ , 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 closely related to the *tree-depth* of  $G$ , which is defined to be the depth of the shallowest DFS tree for a *supergraph* of  $G$ . It is NP-hard to compute the tree-depth even for the class of triangulated graphs [2].

Considering the NP-hardness of the MINKLX problem, the only approach to finding a KLX-minimal, cotranscription-friendly designs for a given graph  $G$  may be via full enumeration of all the DFS trees for  $G$ . In the next section, we present a branch-and-bound approach to this enumeration process.<sup>6</sup>

Before proceeding, let us note that Theorem 5 does not exclude the possibility that KLX-optimal DFS trees could still be found in polynomial time for some graph classes of practical importance such as 3-regular or polyhedral graphs. As any graph can be approximated by a 3-regular one by replacing each vertex by a network of vertices of degree 3, it would be quite interesting to know whether the KLX-minimisation task remains hard for this class. As regards the class of polyhedral graphs, it is known that the MINKLX-related cutwidth minimisation problem (described in the proof of Theorem 5 in Appendix A) is NP-hard for the class of planar graphs with maximum degree 3 [16].

## 6 Solving the MinKLX and MinTD problems by enumeration

Let us now propose a prototype of algorithmic enumeration of DFS trees for computing the KLX number of a given graph  $G = (V, E)$ . Algorithm 2 is its pseudocode. Starting from the empty forest, it enumerates the DFS trees by inserting edges of  $G$  one by one in a predetermined order as a *tree* edge, that is, as an edge of a DFS tree to be built up. Biconnectedness is utilized, the property of a graph being free from an articulation point, whose removal disconnects the graph. DFS trees of a graph cannot circumvent any biconnected components, or *blocks*, of  $G$ . We shall demonstrate how each local branching of a DFS tree orients edges of  $G$  globally once it is accommodated inside a block of  $G$ . Orientations thus insisted by more than one such local branching are highly likely to contradict each other, enabling the incremental enumeration of DFS trees to prune the edge-insertion-based search tree. As a shallower tree involves more branches, the tighter an optional upper bound on the depth of DFS trees to be obtained is set, the more effective this block-based pruning should get.

<sup>6</sup> As an aside, if one gives up the goal of cotranscription-friendly design, a strand routing that minimises the total number of kissing loops needed to complement it can be found efficiently for any graph, and this number is for many typical wireframe models just zero or one [4].



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

---

```

1: Let  $n = |V|$ ,  $m = |E|$ , and edges be indexed as  $E = \{e_1, e_2, \dots, e_m\}$ 
2:
3: function KLXT( $T$ ) ▷ Compute KLX number of tree  $T$ 
4:   compute the KLX number of tree  $T$  in the way described in Section 4
5: end function
6:
7: function KLX( $F, k$ )
8: ▷ Compute min KLX number over all completions of forest  $F$  with edges in  $\{e_k, \dots, e_m\}$ 
9:   if  $F$  comprises a single tree with  $n - 1$  edges then
10:      $klx_{\text{tree}} \leftarrow \text{KLXT}(F)$ 
11:      $klx_{\text{min}} \leftarrow \min\{klx_{\text{min}}, klx_{\text{tree}}\}$ 
12:     return  $klx_{\text{tree}}$ 
13:   end if
14:   if edge  $e_k$  does not create a cycle and is admissible in  $F$  then ▷ See Figure 9
15:      $F' \leftarrow F \cup \{e_k\}$  ▷  $e_k$  included as a tree edge
16:     let  $T$  be the tree that contains edge  $e_k$  in forest  $F'$ 
17:     if  $\text{KLXT}(T) \geq klx_{\text{min}}$  then ▷ Prune if cost of  $T \geq klx_{\text{min}}$ 
18:        $klx_1 \leftarrow m$ 
19:     else
20:        $klx_1 \leftarrow \text{KLX}(F', k + 1)$ 
21:     end if
22:   end if
23:    $klx_0 \leftarrow \text{KLX}(F, k + 1)$  ▷  $e_k$  not included as a tree edge
24:   return  $\min\{klx_1, klx_0\}$ 
25: end function
26:
27:  $klx_{\text{min}} \leftarrow m$ 
28: return  $\text{KLX}(\emptyset, 1)$  ▷ Start with an empty forest

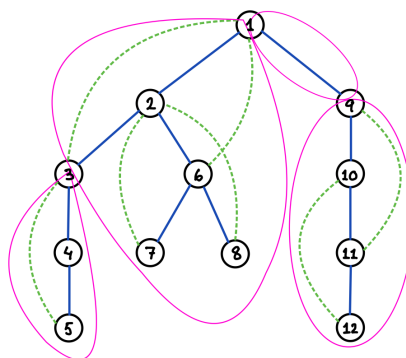
```

---

A graph  $G = (V, E)$  and its spanning tree  $T = (V, S)$  can be uniquely decomposed into a set of biconnected components, or *blocks*,  $E_i \subseteq E$ , along with a spanning tree  $T_i = (V_i, S_i)$  that is the intersection of  $T$  and  $G_i = (V_i, E_i)$ , the induced subgraph of  $G$  by  $E_i$ ; unless confusion arises, we may call even  $G_i$  a block. The blocks can then be uniquely organised into a so-called *block-cut tree* by connecting two blocks by an edge if these blocks induce the subgraphs of  $G$  that share (exactly one) vertex in common, which is an articulation point of  $G$  (see Figure 7 for an example); any articulation point of  $G$  thus serves as an *interface* among multiple blocks. We say that  $G$  is the underlying graph of this block-cut tree. Each block  $G_i$  is one of the following types:

- **Branching.** if  $T$  involves a vertex that is incident to at least three edges in  $E_i$  (*branch*);
- **Spinal.** otherwise.

Every block contains at least one edge of  $T$  and no blocks share an edge; hence, the block-cut tree is composed of at most  $|S| = |V| - 1$  blocks. If a node in a block  $E_i$  is incident to at least two tree edges of the block, say  $e_1, e_2 \in S \cap E_i$ , then we say the node is *internal*. Hence, a spinal block can contain at most two non-internal nodes. Note that an internal node in a spinal block can be a branch of  $T$ ; unlike a branch in a branching block, this branch is an articulation point of  $G$ . Therefore, a spinal block can have three or more interfaces. A spinal block consisting of a single edge is particularly called a *bridge* in [18], and we will borrow this term when it matters whether a spinal block involves an internal vertex or not.



■ **Figure 7** A DFS tree and the biconnected components of a 12-vertex graph.

## 6.1 KLX computation

The decomposition of a graph into blocks may facilitate the computation of the KLX number of  $G$ , but may not yet save it from brute-forcing sibling orders at every branch, which was discussed at the end of Section 4. Consider a block-cut tree that is free from a branching block; note that the underlying graph may admit a DFS tree with a branch because an internal node of a spinal block can serve as an interface as explained above, and three or more spinal blocks may be incident to an articulation point. The absence of branching block enables the KLX number of the underlying graph according to a DFS tree to be computed as the maximum of those of the blocks according to the respective restrictions of the DFS tree; indeed, it makes no sense for the depth-first-search to stay in a spinal block at any internal interface unless all the other blocks incident to the interface have been already traversed.

The computation of the KLX number of each block according to a DFS tree under construction does not require the DFS tree to be rooted but suffices for each block to know at or beyond which interfaces of them lies the global root. As observed shortly, the DFS tree cannot be rooted at any internal node. Hence, a spinal block can be locally rooted only at one of its two non-internal nodes. Its KLX number does not depend on which of these two vertices the global root is on or beyond. Branching blocks rather restrict where the global root can be due to intrinsic orientation of each of their branches, as we see from now on.

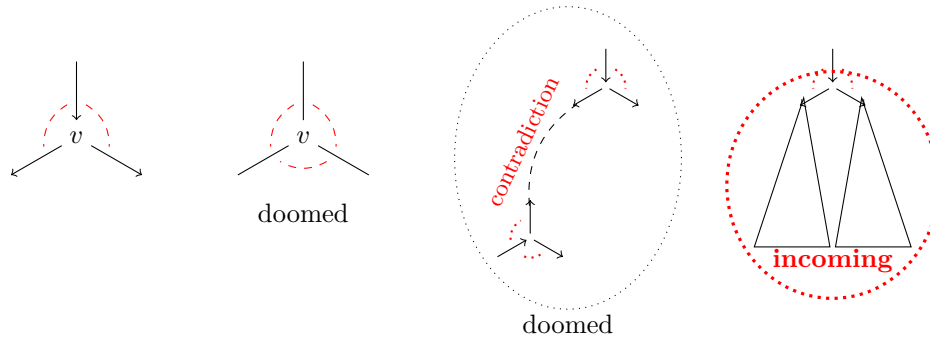
## 6.2 Pruning

The enumerative computation of the KLX number of  $G$  can adopt two kinds of pruning strategies.

The one based on the downward closedness of the KLX number along a specific spanning tree (Lemma 2) is simple; while growing a DFS tree as a block-cut forest, once the KLX number of any tree in the forest exceeds a predetermined target value or the current best, then this path of the enumerative search is not worth being pursued further, and hence should be pruned.

The other strategy is based on the following two structural properties of blocks and their DFS rooting from [13]: given a biconnected graph and its spanning tree like the block  $E_i$  and its spanning tree  $T_i$ ,

1. in order for the spanning tree to be a DFS tree of the graph, it must be rooted at a leaf (of the tree, not of the biconnected graph, all of whose vertices are of degree at least 2);
2. if the spanning tree has a branch, then there exists at most one vertex (indeed, a leaf as just recalled) starting from which the tree can be traversed in a DFS manner.



■ **Figure 8** 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.

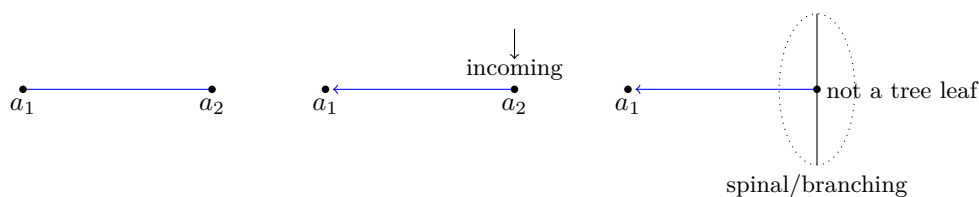
Let us reproduce a proof for Property 2 (Observation 3.2 in [13]). Suppose there were two such vertices  $a$  and  $b$ . If rooted at  $a$ , the branch, say  $v$ , has at least two subtrees  $T_1$  and  $T_2$ , neither of which contains  $a$ ; without loss of generality, we assume that  $T_1$  does not 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 back 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 is rather rooted at  $b$ .

This proof can be applied to observe in our context that every branch inside a branching block is intrinsically oriented, independently of other branches, in such a way that among the edges incident to it, one is incoming while the others are outgoing, as illustrated in Figure 8 (one must be incoming as the branch cannot be a root due to Property 1). Each (local) branch thus globally prohibits a DFS tree of  $G$  from being rooted at any vertex beyond these outgoing edges. As indicated by red dashed lines in Figure 8, a branch cannot be part of a DFS tree of  $G$  if it can be bypassed to go back and forth between any two of its three subtrees (second from the left in the figure). A branch inside a block thus orients some edges across the tree that contains the block, and the in-degree of a vertex is defined naturally. Intrinsic orientations imposed by two branches may contradict each other even inside a block, for example, by yielding a vertex of in-degree 2 or higher (second from the right). As soon as the addition of an edge results in such branch(es), a forest under construction is doomed and should be pruned from the search tree.

### 6.3 Edge insertion and merging of blocks

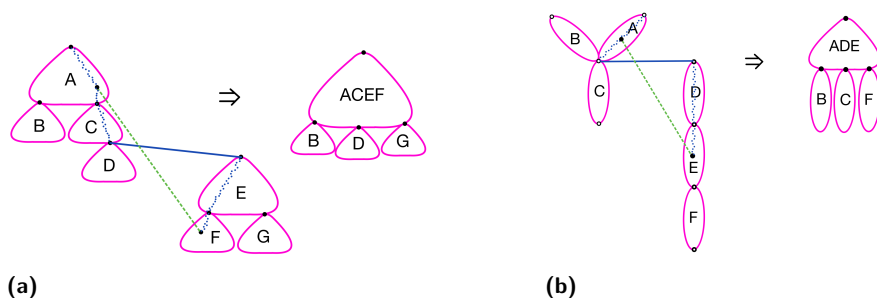
An edge can be added to grow a forest only if all the following conditions hold: (1) it bridges two separate trees, say  $T_1$  and  $T_2$ , (connecting two vertices in the same tree would result in a contradictory cycle of tree edges), (2) at least one of the endpoints, say  $v_1$  and  $v_2$ , is an articulation point of  $G$ , and (3)  $v_1$  or  $v_2$  is of in-degree 0; it is then oriented from the endpoint of in-degree 1, if any, to the other (whose in-degree must be 0); if both endpoints are of in-degree 0, then the added edge remains yet-to-be-oriented (see Figure 9). Let us note, related to (2), that a bridge between an internal node and another node is intrinsically oriented away from the internal node in order to prevent a block from being rooted locally at its internal node. Hence, no tree edge can be added between two internal nodes.

All the other edges between  $T_1$  and  $T_2$  are ruled out as a candidate of tree edges but introduced rather as an ancillary edge. They are however not guaranteed to be consistent; for example, if one of them is between the leaves  $u_1$  and  $u_2$ , another is between the leaves



■ **Figure 9** 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.

$v_1$  and  $v_2$ , and all these four vertices are pairwise distinct, then the tree must be 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 merge some of the blocks in  $T_1$  and in  $T_2$  into one (see Figure 10). The resulting block can be computed efficiently [18] but its KLX number should also be computable more efficiently from those of the merged blocks and from the cost due to the newly added ancillary edges than being computed from scratch.



■ **Figure 10** Two admissible ancillary edge types and their corresponding block-cut tree updates. (a) Edge connecting oriented blocks. (b) Edge connecting unoriented blocks. (Note that in this case the block ADE becomes branching and hence oriented.)

## 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.<sup>7</sup>

Figures 11 and 12 illustrate some outcomes from the tool. Figure 11 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

<sup>7</sup> Design method ST-RNA, additional parameters “co-transcriptional route” and “minimise the number of kissing loop sequences”.

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

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

one. (The spanning tree diagrams in Figures 11(b) and (e)) have been manually reconstructed from the tool-generated diagrams in Figures 11(c) and (f).) Figure 12 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.

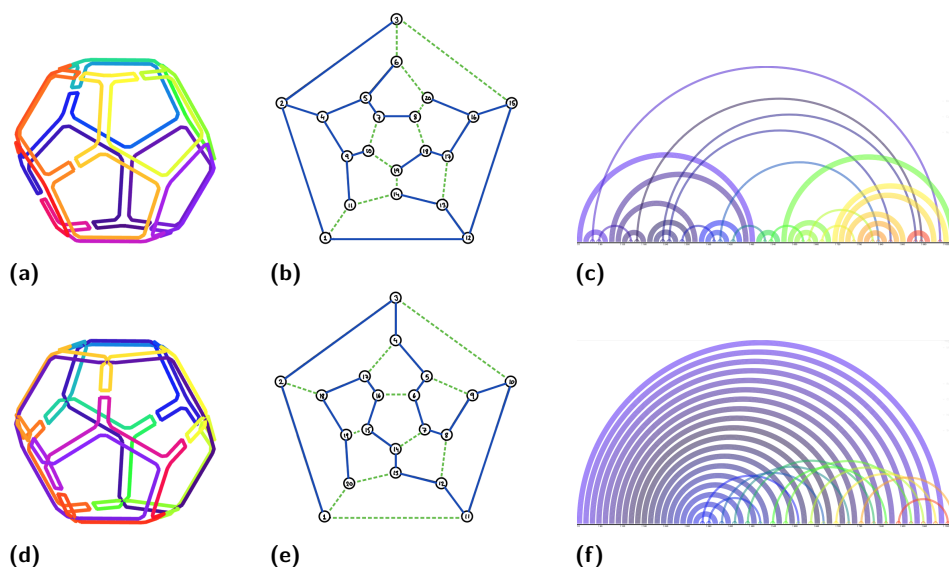
**8** Conclusions and further work

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

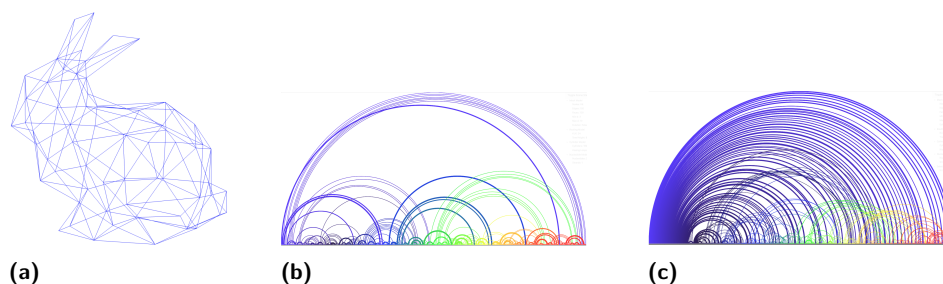
- Relevant directions for further work include for instance the following:
1. Nucleotide-level sequence design for DNA origami wireframes in the cotranscriptional setting.
  2. Efficient combinatorial algorithms for minimising the KLX and TD numbers in some interesting classes of graphs, such 3-regular 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.

**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. doi:10.1016/J.IPL.2005.12.006.
- 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 Antti Elonen and Pekka Orponen. Designing 3D RNA origami nanostructures with a minimum number of kissing loops. In *30th International Conference on DNA Computing and Molecular Programming*. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024. doi:10.4230/LIPICS.DNA.30.4.



■ **Figure 11** 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 green) 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 12** 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).

- 5 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-4102n07t1h>.
- 6 Cody Geary, Guido Grossi, Ewan K. S. McRae, Paul W. K. Rothmund, 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.
- 7 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.
- 8 Cody Geary, Paul W. K. Rothmund, and Ebbe S. Andersen. A single-stranded architecture for cotranscriptional folding of RNA nanostructures. *Science*, 345(6198):799, 2014. doi:10.1126/science.1253920.



- 9 Cody W. Geary and Ebbe Sloth Andersen. Design principles for single-stranded RNA origami structures. In *20th International Conference on DNA Computing and Molecular Programming*, pages 1–19. Springer, 2014. doi:10.1007/978-3-319-11295-4\_1.
- 10 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.
- 11 Peixuan Guo. The emerging field of RNA nanotechnology. *Nature Nanotechnology*, 5(12):833–842, December 2010. doi:10.1038/nnano.2010.231.
- 12 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:10.1021/acsnano.6b05737.
- 13 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.
- 14 Di Liu, Cody W. Geary, Gang Chen, Yaming Shao, Mo Li, Chengde Mao, Ebbe S. Andersen, Joseph A. Piccirilli, Paul W. K. Rothmund, 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.
- 15 Abdulmelik Mohammed, Pekka Orponen, and Sachith Pai. Algorithmic design of co-transcriptionally 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.
- 16 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.
- 17 Erik Poppleton, Niklas Urbanek, Taniya Chakraborty, Alessandra Griffo, Luca Monari, and Kerstin Göpprich. RNA origami: design, simulation and application. *RNA Biology*, 20(1):510–524, 2023. doi:10.1080/15476286.2023.2237719.
- 18 Jeffery Westbrook and Robert E. Tarjan. Maintaining bridge-connected and biconnected components on-line. *Algorithmica*, 7(1):433–464, 1992. doi: 10.1007/BF01758773. doi: 10.1007/BF01758773.

## A NP-completeness of the MinKLX problem

► **Theorem 5.** *The MINKLX problem is NP-hard.*

**Proof.** The proof is based on the proof by Gavril [5] 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 horizontal line in such a way that, for any vertical line drawn between adjacent vertices, dividing  $V$  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  $(\bar{G}, k)$  as follows. With a “large enough”  $r$ , let  $U = \{u_1, u_2, \dots, u_r\}$  be a set of auxiliary “universal” vertices. Let  $\bar{G} = (V \cup U, \bar{E})$  with  $\bar{E} = ((V \cup U) \times (V \cup U)) \setminus E$ . 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  $\bar{G}$  is at most  $k$ .

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

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

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  $u_m, \dots, u_{r-(n-m-1)}$ , 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. ◀