Automated design of dynamic programming schemes for RNA folding with pseudoknots

Bertrand Marchand, Sebastian Will, Sarah Berkemer, Laurent Bulteau, Yann Ponty

Automated design of dynamic programming schemes for RNA folding with pseudoknots

Bertrand Marchand
LIX (UMR 7161), Ecole Polytechnique, Institut Polytechnique de Paris, France
LIGM, CNRS, Univ Gustave Eiffel, F77454 Marne-la-vallée France

Sebastian Will
LIX (UMR 7161), Ecole Polytechnique, Institut Polytechnique de Paris, France

Sarah J. Berkemer
LIX (UMR 7161), Ecole Polytechnique, Institut Polytechnique de Paris, France

Laurent Bulteau
LIGM, CNRS, Univ Gustave Eiffel, F77454 Marne-la-vallée France

Yann Ponty[1]
LIX (UMR 7161), Ecole Polytechnique, Institut Polytechnique de Paris, France

Abstract

Despite being a textbook application of dynamic programming (DP) and routine task in RNA structure analysis, RNA secondary structure prediction remains challenging whenever pseudoknots come into play. To circumvent the NP-hardness of energy minimization in realistic energy models, specialized algorithms have been proposed for restricted conformation classes that capture the most frequently observed configurations.

While these methods rely on hand-crafted DP schemes, we generalize and fully automatize the design of DP pseudoknot prediction algorithms. We formalize the problem of designing DP algorithms for an (infinite) class of conformations, modeled by (a finite number of) fatgraphs, and automatically build DP schemes minimizing their algorithmic complexity. We propose an algorithm for the problem, based on the tree-decomposition of a well-chosen representative structure, which we simplify and reinterpret as a DP scheme. The algorithm is fixed-parameter tractable for the tree-width tw of the fatgraph, and its output represents a $\mathcal{O}(n^{tw+1})$ algorithm for predicting the MFE folding of an RNA of length n.

Our general framework supports general energy models, partition function computations, recursive substructures and partial folding, and could pave the way for algebraic dynamic programming beyond the context-free case.

2012 ACM Subject Classification ; Applied computing → Computational biology; Theory of computation → Dynamic programming

Keywords and phrases RNA folding, treewidth, dynamic programming

Digital Object Identifier 10.4230/LIPIcs.WABI.2022.23

Related Version `https://hal.inria.fr/hal-03676377`

Supplementary Material (implementation) `https://gitlab.inria.fr/bmarchan/auto-dp`

Funding This project has received funding from the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No 101029676, and from the French-Austrian PaRNAssus project (ANR-19-CE45-0023; I 4520-N) supported by the ANR/FWF agencies.

[1] To whom correspondence should be addressed

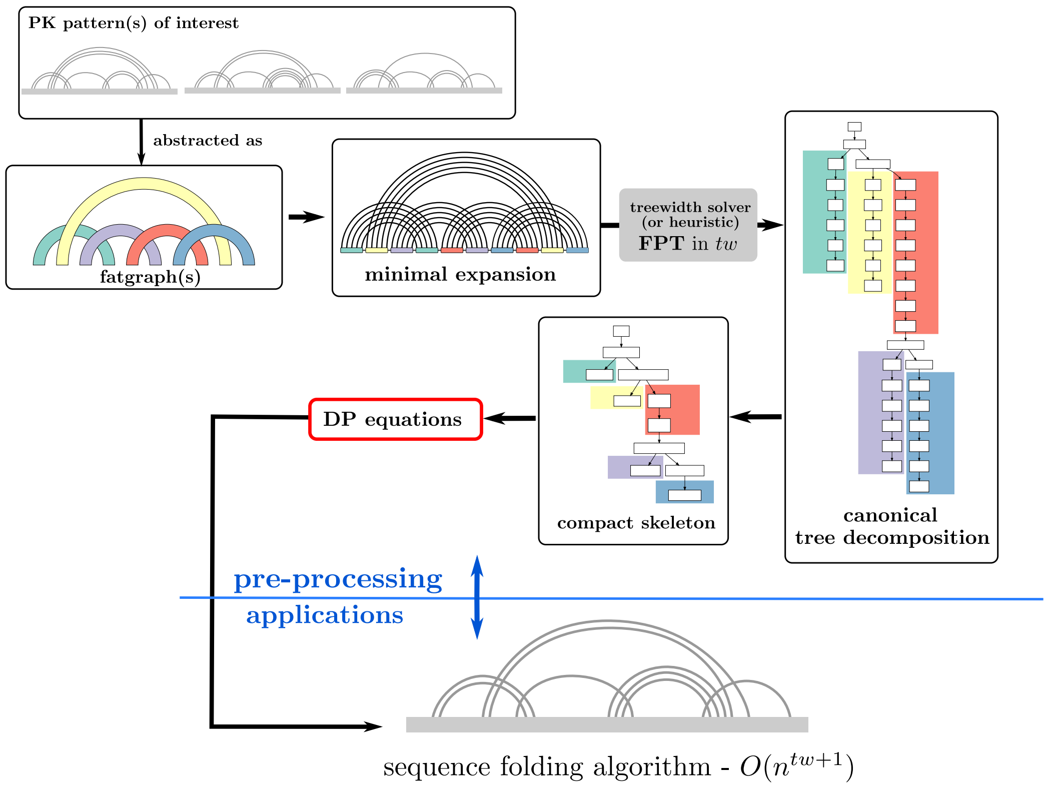

PK pattern(s) of interest
abstracted as
fatgraph(s)
minimal expansion
treewidth solver (or heuristic)
FPT in tw
canonical tree decomposition
compact skeleton
DP equations
pre-processing
applications
sequence folding algorithm - $O(n^{tw+1})$

Figure 1 Given a finite number of arbitrary fatgraphs, a dynamic programming scheme for folding (restricted to the family of structures specified by the fatgraphs) is derived from canonical tree decompositions of minimal representative expansions of the helices, for each fatgraph. The workflow gives an overview of the steps of the algorithm. Each step is described in more details in the subsequent sections and figures: see Figure 2 for fatgraphs, Figure 8 and Section 3 for a detailed version of the canonical tree decomposition, Figure 5 for a detailed view of the compact skeleton of the tree decomposition.

1 Introduction

The function of non-coding RNAs is, to a large extent, determined by their structure. Structure prediction algorithms therefore play a crucial role in (bio-)medical and pharmaceutical applications. The basis to determine more complex 3D structures of RNA molecules is set by first accurately predicting their 2D or secondary structures. There exist various RNA folding algorithms that predict an optimal secondary structure as *minimum free energy structure* of the given RNA sequence in suitable thermodynamic models. In the most frequently used methods, this optimization is performed efficiently by a dynamic programming (DP) algorithm, e.g. `mfold` [47], `RNAfold` [23], `RNAstructure` [36]. A recent alternative to predictions based on experimentally determined energy parameters are machine learning approaches that train models on known secondary structures, e.g., `CONTRAfold` [15], `ContextFold` [46], `MXfold2` [40].

However, the most frequently used algorithms (including all of the above ones) optimize solely over pseudoknot-free structures [44], which do not contain crossing base pairs. Although pseudoknots appear in many RNA secondary structures, they have been omitted by initial prediction algorithms due to their computational complexity [1], and the difficulty to score individual conformations [9]. Nevertheless, many algorithms have been proposed to predict at least certain pseudoknots. These methods are either based on exact DP algorithms such as `pknots-RE` [39], `NUPACK` [14], `gfold` [33], `Knotty` [21] or they use heuristics that don't guarantee exact solutions, e.g., `HotKnots` [35], `IPknot` [41, 40], `Hfold` [20].

Due to the hardness of PK prediction, efficient exact DP algorithms are necessarily restricted to certain categories of pseudoknotted structures. The underlying DP schemes are designed manually, guided by design to either i) support structures that are frequently observed in experimentally resolved structures (declarative categories); or ii) support the largest possible set of conformations, while remaining within a certain complexity (complexity-driven). For most categories, essentially declarative ones, there exists one or several helix arrangements, either observed in experimentally-determined structures or implicitly characterized by graph-theoretical properties (3 non-crossing [34], topologically bounded [33]) that need to be captured. A detailed overview of pseudoknot categories is given in [27]. Similar situations occur for RNA-RNA interactions [2], possibly including several RNA molecules. Interestingly, when more than two RNA strands are considered, existing algorithms restrict the joint conformation to crossing-free interactions [16], further motivating the design of algorithms beyond the case of pseudoknot-free secondary structures.

In this work, we describe classes of pseudoknotted structures as fatgraphs [19, 33, 22, 30], an abstraction of RNA conformations related to RNA shapes [17] or shadows [34, 33]. We formalize the principles underlying the design of DP folding algorithms including pseudoknots, and, at the same time, give a formulation of the computational problem based on the design of DP algorithms. We show how to leverage tree-decompositions, computed on a minimal expansion of the input fatgraph, to automatically derive DP schemes that use as little indices as possible. Our algorithm can be interpreted as a generalization of the algorithms underlying `LiCoRNA` [38] and `gfold` [33] and we propose a parameterized algorithm based on the treewidth (tw) of the underlying fatgraph.

In Section 2, we state our problem and define its input structure abstraction, the fatgraph. Then, we describe helix expansions of the fatgraph and their tree decompositions (Section 3). By minimal helix expansions and a derivation of the tree decomposition to its canonical form, we automatically derive a DP scheme for the folding of pseudoknotted structures (Section 4), using a number of indices equal to the treewidth. Figure 1 outlines the fundamental algorithm. Section 5, discusses extensions to combine multiple fatgraphs, include recursive substructure, and cover realistic energy models.

2 Definitions and main result

We define an *RNA sequence* S as a word of length n over the nucleotides A, C, G and U; moreover an *RNA secondary structure* (potentially, with pseudoknots) ω of S as a set of *base pairs* (i, j) between sequence positions i and j (in $1, ..., n$), such that there is at most one base pair incident to each position. A *diagram* is a graph of nodes 1,...,n (the positions), connecting consecutive positions by directed edges $(i, i+1)$ and moreover connecting positions by arcs, visualizing the *arc-annotation* of the sequence. Typically this is represented drawing the backbone linearly and the arcs on top. RNA secondary structures are naturally interpreted as diagrams.

One of our central concerns is the crossing configuration of arcs in a diagram. We define two arcs (i, j) and (i', j') in a diagram as *crossing* iff $i < i' < j < j'$ or $i' < i < j' < j$. Naturally, this leads to the notion of a conflict graph consisting of all the arcs of a diagram

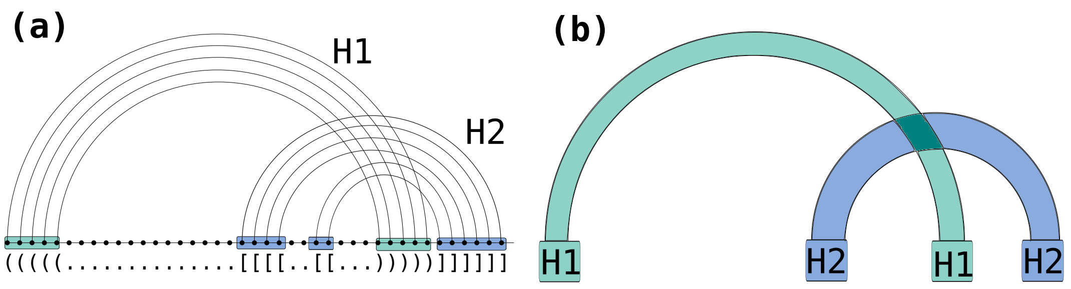


Figure 2 (a) Diagram of a secondary structure with two crossing helices (H1 green, H2 blue). (b) fatgraph corresponding to the above structure such that helices are collapsed into bands and form the shadow of the structure.

and connecting crossing arcs by a conflict edge. Given a potentially conflicted set of base pairs, the associated *RNA structure graph* is the diagram consisting of one vertex per nucleotide, backbone links, and one arc per base pair.

A *fatgraph* [19, 33, 22, 30] is an abstraction of a family of pseudoknotted RNA structures displaying a specific conflict structure. It is typically represented as a *band diagram* (see Figure 1 and Figure 2), in which each band may represent a *helix* of arbitrary size, including bulges. An arc-annotation is said to be an *expansion* of a fatgraph if collapsing nested arcs and contracting isolated bases yields the band diagram of a fatgraph. Given a finite number of fatgraphs, we say a structure is a *recursive expansion* of these fatgraphs if decomposing the structure into conflict-connected components, collapsing nested arcs and contracting isolated bases only yields members of the given fatgraph set. For the purpose of this presentation (where we do not explicitly study structure topology), we moreover identify fatgraphs with their diagrams.

To make the connection to `gfold` [33] explicit, recursive expansions of fatgraphs are equivalently understood in terms of the shadows of a structure. The shadow of an RNA structure (or equivalently, its diagram) is defined in [33] as the diagram obtained by, firstly, removing all unpaired bases and non-crossing structures and, secondly, contracting all stacks (i.e. pairs of arcs between directly consecutive positions) to single arcs. Then, the class of recursive expansions of a set of input fatgraphs Γ is the class of structures, where the shadows of their conflict-connected components are in Γ.

In this paper, we consider a class of RNA folding problems in which the search space is restricted to recursive expansions of a user-specified finite set of fatgraphs. For the sake of simplicity, we first describe minimizing energy in a simple free-energy model $\mathcal{E}$, where the energy of a sequence/structure is obtained by summing the contributions of individual base pairs; moreover, we present the method initially without recursive substructure. Only later, in Section 5, we extend to the full problem in realistic energy models.

▶ **Definition 1** ((Recursive) fatgraph MFE folding problem).
Input: *Finite collection of fatgraphs $\gamma_1, \ldots, \gamma_p$, sequence S*
Output: *Minimum Free Energy (MFE) arc-annotation for S according to free-energy model $\mathcal{E}$, restricting the search to recursive expansions of the input fatgraphs.*

Specifically, we solve the problem of automatic design of such pseudoknot prediction algorithms based on an input set of fatgraphs.

▶ **Definition 2** (Fatgraph folding algorithm design problem).
Input: *Finite collection of fatgraphs $\gamma_1, \ldots, \gamma_p$, sequence S*

ALGORITHM 1 Pseudocode for the recursive fatgraph folding problem.

```
Input  : Finite number of fatgraphs $\gamma_1, \ldots, \gamma_p$, sequence $S$, base-pair based energy model $\mathcal{E}$
Output : Best-scoring arc-annotation for $S$, in the class specified by the fatgraphs
1 foreach fatgraph $\gamma_i$ do
2     Compute minimal expansion $G_i$ of fatgraph $\gamma_i$        ▶ Linear time; see Section 3.2
3     Find min. width tree decomposition $\mathcal{T}$ for $G_i$   ▶ FPT in $tw$ using classic exact tree dec. algorithm
4     Transform $\mathcal{T}$ into a canonical form tree dec $\mathcal{T}'$   ▶ Polynomial time; see Section 4.1
5     Compute skeleton of $\mathcal{T}'$                             ▶ Linear time; see Section 4.1
6     Derive corresponding DP scheme                                 ▶ Linear time; see Section 4.2
7 end
8 Use union of DP schemes to find MFE arc-annotation of $S$  ▶ XP in tw $O(n^{tw+1})$; See Section 5
```

Output: *A Dynamic-Programming algorithm that efficiently returns the MFE arc-annotation for S, with respect to free-energy model $\mathcal{E}$, over the recursive expansions of the input fatgraphs.*

Defining the treewidth of a fatgraph as the treewidth of its minimal expansion (see Section 3.2), our main result, stated in Algorithm 1, is the existence of an effective algorithm for the fatgraph-folding problem, XP over *tw* the maximum treewidth of the input fatgraphs. Its first step consists in a Fixed-Parameter Tractable (FPT) pre-processing of the input fat graphs, yielding DP equations for folding (see Figure 1), which can be reused to fold any other input sequence.

▶ **Theorem 3** (Main result). *Algorithm 1 solves the fatgraph folding problem in $O(n^{tw+1})$, where tw is the maximum treewidth of the input fatgraphs.*

Since the number of indices used by the DP equation is minimized, the resulting complexities could be seen as optimal within a family of simple DP algorithms. However, a characterization of such a non-trivial family of algorithms would be beyond the scope of this work, and we leave formal proofs of optimality to future work, as briefly discussed in Section 7.

3 Minimal representative expansion of a fatgraph

Our approach builds on the concept of tree decomposition, which we want to leverage to derive decomposition strategies within dynamic programming (DP) schemes. A key challenge is in the fact that tree decompositions are computed for concrete graphs, whereas our objective is to find an algorithm whose search space includes all possible expansions of an input fatgraph.

Fortunately, we find that expanding every helix of a fatgraph to length 5 (i.e. 5 nested BPs) yields a graph which is representative of the fatgraph. Namely, its optimal *tree decomposition*, having *treewidth tw*, trivially generalizes into a tree decomposition for any further expansion, retaining *treewidth tw*. This tree decomposition can finally be reinterpreted into a DP scheme that exactly solves the MFE folding problem in $\mathcal{O}(n^{tw+1})$ complexity.

3.1 Treewidth and tree decompositions

▶ **Definition 4.** *A tree decomposition $\mathcal{T} = (T, \{X_i\}_{i\in V(T)})$ of a graph $G = (V, E)$ is a tree of subsets of vertices of G, called bags, verifying the following conditions:*

- $\forall u \in V\ \exists i \in V(T)$ *such that* $u \in X_i$. *(representing vertices)*
- $\forall (u, v) \in E\ \exists i \in V(T)$ *such that* $\{u, v\} \subset X_i$. *(representing edges)*
- $T_u = \{i \in V(T) \mid u \in X_i\}$ *must be connected.* *(vertex subtree property)*

The *width* of a tree decomposition is the size of its biggest bag minus one, i.e. $\max_{i\in V(T)} |X_i| - 1$. The *treewidth* of a graph G is then the minimum possible width of a tree decomposition of G. Intuitively, the lower the treewidth, the closer G is to being a tree. Treewidth is NP-HARD to compute [3], but fixed-parameter tractable: there is a $O(f(w)\cdot n)$ algorithm [5] deciding whether $tw(G) \leq w$ given G. Many polynomial heuristics are also known to yield reasonable results [8], and optimized exact solvers have also been developed [43, 18]. Notoriously, a wide variety of hard computational problems can be solved efficiently when restricted to graphs of bounded treewidth [7, 11], including in bioinfomatics [45, 42, 38]. Such is the case of LiCoRNA [38], for pseudoknotted structure-sequence alignment, of which the algorithm presented in this paper can be seen as a generalization.

We will rely in the remainder of this section on some well known-properties for treewidth, which we recall here. First, taking any *minor* of G [24], i.e. performing any sequence or edge contractions, edge deletions and vertex deletions on G can only lower the treewidth. Second, degree-2 vertices can be contracted into their neighbors without changing the treewidth, as quickly stated below (proof in appendix). This implies in particular that any bulge in a helix of an RNA structure graph is inconsequential with respect to treewidth.

▶ **Proposition 5.** *If u is a degree-2 vertex of G with neighbors $\{v,w\}$, and $G_{v\leftarrow u}$ is the graph obtained by contracting u into v in G then $tw(G) = tw(G_{v\leftarrow u})$*

Then, we import from [6] an inequality valid for any *separator* of G. A *separator* is a subset S of vertices of G such that $G \setminus S$ is composed of at least 2 conected components, which we write $\mathcal{C}_G(S)$. We then have:

▶ **Proposition 6.** *If S is a separator of G, then*

$$tw(G) \leq \max_{C\in\mathcal{C}_G(S)} tw(G[C \cup clique(S)])$$

with $G[C \cup clique(S)]$ the subgraph of G induced by $C \cup S$ augmented by edges making S a clique. In case of equality, we say that S is safe.

Proof. Consider, for each $C \in \mathcal{C}_G(S)$, a tree decomposition $\mathcal{T}_C$ of $G[C \cup clique(S)]$. Since these graphs contain S as a clique, each $\mathcal{T}_C$ must have a bag X_C containing S entirely. Consider now the following tree decomposition for G, make a bag out of S, and connect X_C for each C to it. The resulting tree decomposition is valid for G, and its width is the left-hand-side of the inequality. ◀

Let us finish by noting that, in a tree decomposition, any intersection $S = X \cap Y$ of two adjacent bags is always a separator of G. To write down the proofs of the following section in a smoother fashion, we add the following two properties, whose proofs are delayed to the appendix:

▶ **Proposition 7.** *A tree decomposition can always be locally modified such that, for any two adjacent bags X and Y and $S = X \cap Y$:*

- *$|S| \leq tw(G)$*
- *S is minimal with respect to inclusion, i.e. removing any vertex from S makes it lose its separating properties.*

3.2 Helices of length 5 are sufficient to obtain generalizable tree decompositions

Given an RNA graph (with one vertex per nucleotide and one edge per base pair and backbone link, see Figure 3(a)), we call *perfect helix* a set of directly nested base pairs, resulting in the subgraph depicted on Figure 3(b). We call the number of nested base pairs its *length*, and denote it with l. With a slight abuse of language, we call such a subgraph a *helix*, even for general graphs. Our main structural result is to show that the treewidth of a graph G does not increase when extending a helix past a length of 5. Its proof relies on the following inequality, involving the graphs $G_{\boxtimes}$ and $G_{\boxslash}$, obtained from G by replacing a helix H with either $\boxtimes$ or $\boxslash$, (see Figure 3(c)).

▶ **Lemma 8.** *Given a graph G and a helix H of length $l \geq 3$ in G, we have:*

$$tw(G_{\boxtimes}) - 1 \leq tw(G_{\boxslash}) \leq tw(G) \leq \max(4, tw\,(G_{\boxtimes}))$$

Proof. To start with, by noticing that the 4 extremities of the helix form a separator S between the inside and the outside of it, we get by Proposition 6 that $tw(G) \leq \max(H \cup clique(S), G_{\boxtimes})$. The graph $H \cup clique(S)$ does not depend on G, and consists of a helix with the 4 extremities forming a clique. With $l \geq 2$, it turns out that this graph has treewidth 4, see Appendix A, hence the inequality.

Next, we notice that $G_{\boxslash}$ is a minor of G when $l \geq 3$. This can be seen by contracting the helix according to the pattern outlined on Figure 3(d) by the green areas (each green area is contracted to the extremity it contains). Therefore, $tw(G_{\boxslash}) \leq tw(G)$.

Finally, let us note that $G_{\boxtimes}$ and $G_{\boxslash}$ only differ by 1 edge, and removing a single edge from a graph can only decrease its treewidth by at most 1. Indeed, suppose that $tw(G_{\boxslash}) < tw(G_{\boxtimes}) - 1$, and consider an optimal tree decomposition $\mathcal{T}$ for $G_{\boxslash}$. Let us denote by u and v the two extremities of the helix not connected in $G_{\boxslash}$. If the subtrees of bags containing respectively u and v do not intersect, then one can just add v to all bags of the tree decomposition, to represent the edge (u, v) while increasing the width by ≤ 1. Therefore $tw(G_{\boxtimes}) - 1 \leq tw(G_{\boxslash})$ and the inequality is complete. ◀

Through the introduction of $G_{\boxtimes}$ and $G_{\boxslash}$ as the two possible graphs to which G is equivalent in terms of treewidth, Lemma 8 already contains the essence of our main structural result, Theorem 9. It will be the basis for generalizing tree decompositions of minimal expansions of a fatgraph to arbitrary helix lengths. Its proof is delayed to Appendix E.

▶ **Theorem 9.** *If H is a helix in G of length $l \geq 5$, then extending the helix to have length $l + 1$ does not increase the treewidth.*

Since bulges in a helix only consist of vertices of degree exactly 2, combining Proposition 5 with Theorem 9 implies that the treewidth of any expansion of a given fatgraph is always smaller than or equal to the treewidth of a minimal expansion where all bands are helices of length exactly 5. As for gaps, arguments similar to the proof of Theorem 9 can show that going from a gap of length 0 to an arbitrary length does not increase the treewidth of a fatgraph expansion. Overall, we formally define the minimal expansion of a fatgraph as:

▶ **Definition 10** (Minimal representative expansion of a fatgraph)**.** *Given a fatgraph γ, its minimal representative expansion consists of:*

- *A perfect helix of length 5 for each band.*
- *No gap between the extremities of two helices*

Such a minimal representative expansion is illustrated in Figure 5(a). For visual clarity, gaps have been kept between consecutive helices, but one can see that the corresponding extremities have the same labels. Given a fatgraph, this RNA structure graph contains all necessary information for formulating DP equations decomposing all RNA structures compatible with the fatgraph.

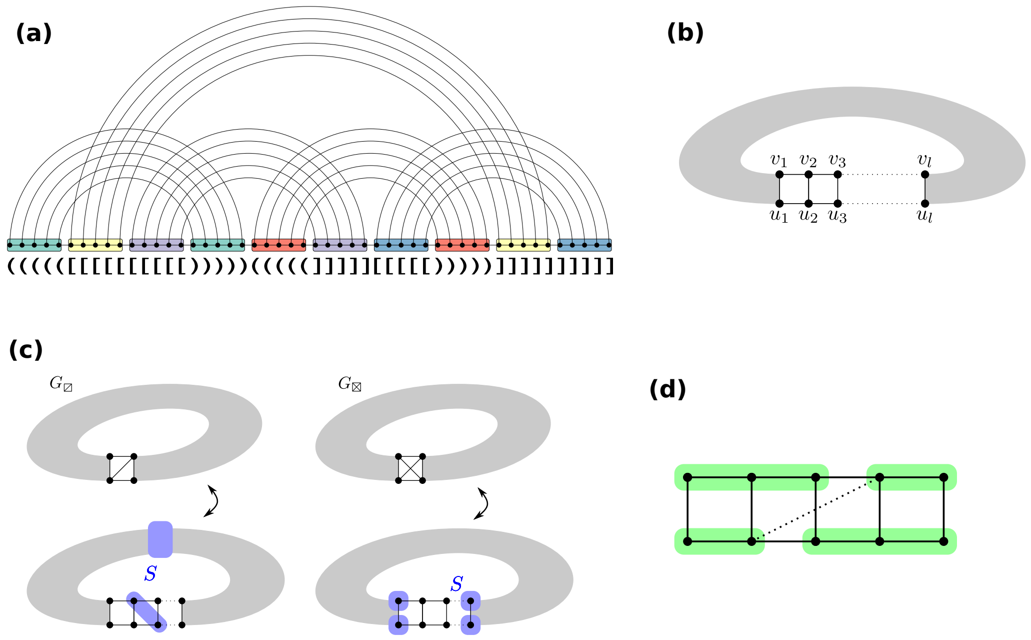


Figure 3 (a) minimal expansion of a fatgraph, with every helix of length 5, and no unpaired base. The associated graph consists of one vertex per base, and one edge per base pair and backbone link. (b) A helix of length l in an RNA graph, as per the latter definition. (c) Given a helix in a graph G, the treewidth of G is either equal to $tw(G_{\boxtimes})$ or $tw(G_{\boxslash})$. Each case is associated with a type of *separator* that can be used to extend the helix, or insert bulges, without changing the treewidth. (d) The dotted line represents a "hop-edge" which, if represented in a given tree decomposition of G, can be used to obtain $G_{\boxtimes}$ as a minor of G, showing that the helix is in the "clique" case.

Interestingly, the two graphs $G_{\boxtimes}$ and $G_{\boxslash}$ that emerge in the proofs as the two graphs G could be equivalent in terms of treewidth, as well as the separators they are associated to (see Figure 3 (c)) are reminiscent of two typical decomposition strategies used into dynamic programming for RNA folding. They suggest, for each helix in a graph, two possible "canonical representations" in terms of tree decomposition, which will be elaborated on in the next section.

4 Tree decompositions of fatgraph expansions as RNA DP algorithms

Starting with a tree decomposition for a minimal representative expansion of a given fatgraph, we first describe in this section how to represent it in a *canonical form*, with each helix represented either in one of two different ways, respectively related to $G_{\boxslash}$ and $G_{\boxtimes}$. The resulting tree decomposition can be further compressed into a *skeleton*, where bags within individual helices are compressed into a single bag.

This tree can then be interpreted as a dynamic programming scheme, in which helices are generated by specializing dynamic programming subroutines. In a sense, the tree decomposition yields automatically a decomposition strategy usable for dynamic programming, of the kind that was hand-crafted in previous approaches [33, 14].

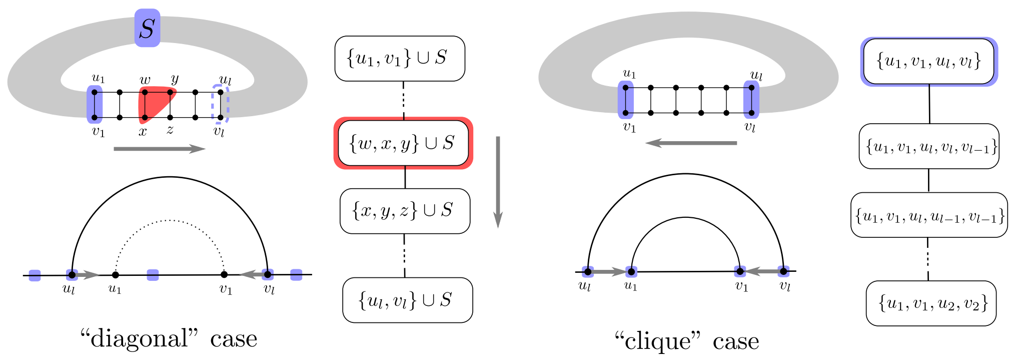


Figure 4 Illustration of the two types of canonical representations for the helices of a graph G

4.1 Canonical form for tree decompositions

We introduce an additional definition for the sake of presentation: Given an edge $e = (X, Y)$ of a tree decomposition $\mathcal{T}$, we call the $X - side$ of $\mathcal{T}$ the connected component of $T \setminus e$ containing X.

▶ **Definition 11.** *A tree decomposition of an expansion G of a fatgraph is in canonical form if, for each helix H of length l, either:*

- ***Clique case:*** *Helix H is represented by a root bag that contains all 4 extremities of H, connected to a sub-tree-decomposition T_l recursively defined as*

$$T_0^{\boxtimes} = \emptyset \quad T_l^{\boxtimes} = \{u_1, v_1, u_l, v_l\} \to \{u_1, v_1, u_l, v_{l-1}, v_l\} \to \{u_1, v_1, u_{l-1}, u_l, v_{l-1}\} \to T_{l-1}^{\boxtimes}.$$

- ***Diagonal case:*** *Helix H is represented by a linear series of bags starting with $X_1 = S^* \cup \{u_1, v_1\}$, finishing with $X_{2l+2} = S^* \cup \{u_l, v_l\}$, and such that for $1 < k < l + 1$ $X_{2k} = S^* \cup \{u_{2k-1}, v_{2k-1}, u_{2k}\}$ and $X_{2k+1} = S^* \cup \{v_{2k-1}, u_{2k}, v_{2k}\}$ for k odd.*

The definition above is illustrated on Figure 4. A canonical tree decomposition for a minimum expansion of a fatgraph is also presented on Figure 8. It was obtained through the processing routine that we describe in Algorithm 2 (see Appendix D), applicable to any (optimal or not) tree decomposition. It essentially follows the dichotomy of the proof of Theorem 9. We state its correctness and run-time below, but delay the proof to Appendix E.

▶ **Proposition 12.** *Given G and $\mathcal{T}$, Algorithm 2 outputs a canonical tree decomposition for G, having same width as T, in time $O(N_H \cdot n^3)$, where N_H is the number of helices.*

Note that in a canonical tree decomposition, all vertices and edges internal to a helix of a graph are represented in the canonical sub-tree-decomposition associated to it. All bags outside of these canonical blocks only consist of extremities of helices, or other vertices outside of helices. Ignoring these internal parts, to focus on a more compact "skeleton" of canonical tree decompositions will be the first step towards automatically deriving dynamic programming equations.

▶ **Definition 13.** *The skeleton of a canonical tree decomposition for a graph G, is defined as follows:*

- *All sub-tree-decompositions representing a helix in the "clique" case are replaced with a unique bag containing all extremities of the helix*

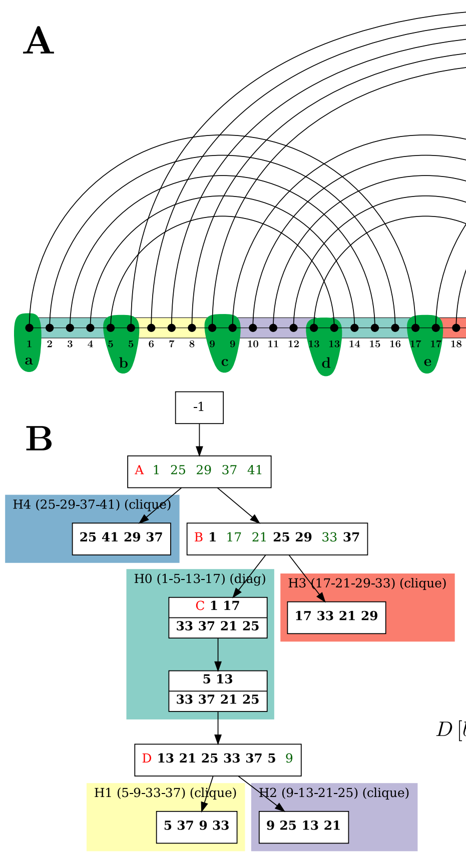
- *All sub-tree-decompositions representing a helix in the "diagonal" case are contracted to contain their first and last bags only, denoted as $S \cup \{u_1, v_1\}$ and $S \cup \{u_l, v_l\}$ in Definition 11.*

Figure 5(b) gives an example of such a skeleton.

4.2 Automatic derivation of dynamic programming equations

Given the skeleton of a representative minimal expansion of a fatgraph γ, we describe here how to formulate DP equations for the corresponding folding problem. As mentioned previously, we initially restrict our exposition to a base-pair based model, akin to the one optimized by the seminal Nussinov algorithm [29].

Essentially, we introduce helix DP tables for each helix, and transitional tables for non-helix bags. The variables indexing these tables are called *anchors*. These integer variables



C

$$A = \min_{a,g,h,j,k} \left(B\,[a,g,h,j] + \boxed{C_{\boxtimes}}[g,h-1,j,k-1]\right)$$

$$B\,[a,g,h,j] = \min_{e,f,i} \left(\boxed{C_{\boxtimes}}[e,f-1,h,i-1] + \boxed{C}\,[a,e|f,g,i,j]\right)$$

$$\boxed{C}\,[a,e|f,g,i,j] = \min \begin{cases} C[a+1,e|f,g,i,j], \\ C[a,e-1|f,g,i,j], \\ C[a+1,e-1|f,g,i,j] + \Delta G(a,e), \\ D[a,e+1,f,g,i,j] \end{cases}$$

$$D\,[b,d,f,g,i,j] = \min_{c} \left(\boxed{C_{\boxtimes}}[c,d-1,f,g-1] + \boxed{C_{\boxtimes}}[b,c-1,i,j-1]\right)$$

Figure 5 (a) Minimal representative length-5 expansion of the fat graph shown in Figure 1. Anchor variables are highlighted in green. We introduce one such variable per gap between helices. (b) Skeleton of the tree decomposition. White boxes represent transitional bags, introducing/propagating indices, while colored boxes represent helices in the fatgraph (H0 to H4) with associated indices in the input structure. Red letters indicate tables of the dynamic programming algorithm. Green indices are novel indices, absent from a bag's predecessor. (c) DP equations derived from the compact skeleton, involving the anchor variable defined above, and following the rules described in Section 4.2.

each represent a separation point between consecutive (half-)helices. Taken together, a full set of anchors $(a, b, c, \ldots)$ partitions the sequence into a set of disjoint intervals $[a, b[, [b, c[\ldots$, each associated with one *half-helix*, i.e. one of the subsequences that form a helix. Helix tables will account for the free-energy contributions of concrete base-pairs, while transitional tables will instantiate anchors in a way that remains consistent with previous assignments.

Indeed, owing to the tree decomposition, a skeleton is guaranteed to: i) feature each anchor in some bag; ii) represent each pair of consecutive anchors in at least one bag; iii) propagate anchor values, such that the anchor values within helix tables remain consistent. Due to this observation, non-helix bags can simply propagate previously-assigned anchors, possibly assigning values to novel anchors (if any and constrained to remain consistent with the sequential order) to explore all possible partitions of the input RNA sequence.

Helix tables will predict concrete sets of base pairs and account for their associated free-energy. In order to both prevent the double pairing of certain sequence positions, and to avoid ambiguity, we require (and enforce in the DP rules) that an anchor x, separating the consecutive halves of two helices H and H', implies the pairing of position x to the other half of H', and the pairing of some position $x' < x$ as part of H. In other words, a helix H delimited by anchors i, i', j', and j must pair position i to some position $x \in]j', j[$, and j' to some position $y \in]i, i'[$, implicitly leaving both regions $]y, i'[$ and $]x, j[$ unpaired.

4.2.1 Helix table 1: "Clique" cases

In the skeleton, each bag representing a helix in the "clique" case is associated to the following tables, where $i, i'+1, j'$, and $j+1$ represent the values of the anchors delimiting the helix. The increments on i' and j are here to ensure the presence of gap of length ≥ 1 between two base pairs belonging to different helices. (see also Figure 5(c) for an example of how anchor values are passed to $C_{\boxtimes}$ with a decrement of -1 for the same reason).

A first table $C'_{\boxtimes}$ holds the minimal free-energy of a helix delimited by i, i', j', and j, such that position i is paired to some $x \in]j', j[$ and j' to some position $y \in]i, i'[$. The idea is here to iteratively move the anchor from j to $j-1$, implicitly leaving position j unpaired, until a base pair (i, j) is formed. Once a base pair is created, we transition to another table $C_{\boxtimes}$ which optimizes over helices like $C'_{\boxtimes}$, but additionally allows position i to be left unpaired.

Those two tables can be filled owing to the following recurrences:

$$C'_{\boxtimes}[i, i', j', j] = \min \begin{cases} C'_{\boxtimes}[i, i', j', j-1] & \text{if } j' < j \\ C_{\boxtimes}[i+1, i', j', j-1] + \Delta G_{i,j} & \text{if } (i < i') \wedge (j' < j) \\ \Delta G_{i,j} & \text{if } j = j' \\ +\infty & \text{if no such case apply} \end{cases}$$

and

$$C_{\boxtimes}[i, i', j', j] = \min \begin{cases} C'_{\boxtimes}[i, i', j', j-1] & \text{if } j' < j \\ C_{\boxtimes}[i+1, i', j', j] & \text{if } i < i' \\ C_{\boxtimes}[i+1, i', j', j-1] + \Delta G_{i,j} & \text{if } (i < i') \wedge (j' < j) \\ \Delta G_{i,j} & \text{if } j = j' \\ +\infty & \text{if no such case apply} \end{cases}$$

where $\Delta G_{i,j}$ denote the free-energy contribution of the base-pair (i, j) in the input RNA sequence.

4.2.2 Helix tables 2: "Diagonal" cases

In the skeleton bags representing the diagonal cases, we need to associate a different table to each helix. Indeed, each "diagonal" case associates, to a helix H, a set S of indices, dubbed the *constant anchors*, whose values remain unchanged during the construction of H.

We focus on the case where (i, j) represents the value of the outermost anchor pair (i.e. $[i, j]$ represents the full span of H), leaving to the reader the symmetric case starting from the innermost pair. Note that, in the skeleton, we kept two bags for a "diagonal case" helix. Yet they are associated to a single table, since the helix is created by incrementing two indices only, such that the initial pair of extremities "becomes" the other pair. We need this second bag to know how to map index values to the children tables $\{M_k\}_k$. This value mapping at the end of a diagonal case is illustrated on Figure 6.

Namely, let the cell $D_H[i, j \mid S]$ (resp. $D'_H[i, j \mid S]$) represent the minimum-free energy achieved by the set of helices in the subtree of H, when H is anchored at (i, j) without constraints on i or j (resp. such that i is paired to some position $x \leq j'$). We have:

$$D'_H[i,j \mid S] = \min \begin{cases} D'_H[i,j-1 \mid S] & \text{if } j-1 > i \wedge \forall s \in S,\ j-1 \neq s \\ D_H[i+1,j-1 \mid S] + \Delta G_{i,j} & \text{if } \forall s \in S,\ (i+1 \neq s) \wedge (j-1 \neq s) \end{cases}$$

and

$$D_H[i,j \mid S] = \min \begin{cases} D_H[i+1,j \mid S] & \text{if } i+1 < j \wedge \forall s \in S,\ i+1 \neq s \\ D'_H[i,j-1 \mid S] & \text{if } j-1 > i \wedge \forall s \in S,\ j-1 \neq s \\ D_H[i+1,j-1 \mid S] + \Delta G_{i,j} & \text{if } \forall s \in S,\ (i+1 \neq s) \wedge (j-1 \neq s) \\ \sum_k M_k[I_k] & \text{with } I_k := (\{i, j+1\} \cup S) \cap A_k \end{cases}$$

where A_k denotes the anchors values needed for the k-th child of the diagonal bag.

4.2.3 Transitional tables: Non-helix bags

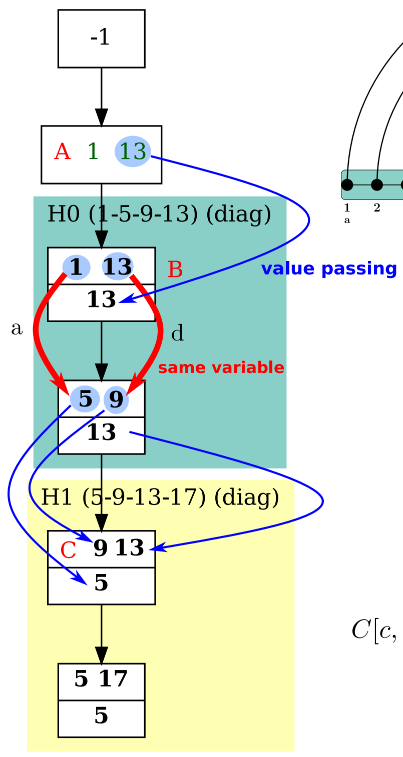
The general case consists of passing the values of relevant variables onward to the diagonal and clique tables, possibly assigning/propagating anchors that appear in the bag for the first time. Let I_P be the anchors of the parent bag of M in the tree decomposition, we have:

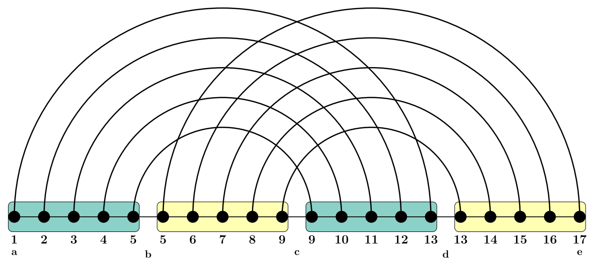
$$M[I_P] = \min_{\substack{\text{Values for all} \\ \text{anchors in } I \backslash I_P}} \sum_k \begin{cases} M_k[I_k] & \text{if } k\text{-th child is transitional} \\ C'_{\boxtimes}[i, i'-1, j', j-1] & \text{if clique, anchored at } (i, i', j', j) \\ D'_{H_k}[i, j-1 \mid S_k] & \text{if diagonal, anchored at } (i, j') \end{cases}$$

where I_k denotes the anchor values from I needed for the k-th child of the bag, and S represents the constant anchors of the k-th child, assumed to be a diagonal.

5 Extensions

The DP scheme, as stated above, only supports conformations that consist of a single pseudoknot configuration, indicated by a fatgraph. Moreover, it forces the first position of the sequence to always form a base pair. Finally, it considers an energy model that is fairly unrealistic in comparison with the current state of the art. In this section, we briefly describe how to extend this fundamental construction in several directions in order to solve the stated algorithm design problem (Def. 2) and consequently the associated folding problem in complex energy models, and discuss the consequences on the complexity.





$$A = \min_{a \leq d} B[a, d|d]$$

$$\downarrow$$

$$B[a, d|d'] = \min\,(B[a+1, d|d'],\\ B[a, d-1|d'],\\ B[a+1, d-1|d'] + bp(a, d),\\ C[d, d'|a])$$

$$\downarrow$$

$$C[c, d|b] = \min\,(C[c-1, d|b],\\ C[c, d+1|b],\\ C[c-1, d+1|b] + bp(c, d))$$

Figure 6 Derivation of DP equations from a skeleton, starting from the canonical tree decomposition of a length-5 expansion for a simple H-type fatgraph. On the left-hand-side, special emphasis is given to explaining how values are mapped at the end of a diagonal case. Extra tables $C'_{\boxtimes}$ and D'_H, needed to ensure unambiguity of the DP scheme, are omitted for the sake of simplicity without adverse consequences to correctness.

5.1 Multiple fatgraphs and integration within 2D folding scheme

Alternative fatgraphs can easily be considered, without added complexity, by simply adding a disjunctive rule at the top level of the DP scheme, such as $\text{MFE}_{\text{PK}} := \min_{i=1}^{p} root_{\gamma_i}$ where $root_{\gamma_i}$ is the top level of the DP scheme for fatgraph γ_i. The associated conformation space then consists of the union of all pseudoknotted structures compatible with one of the fatgraphs.

However, fatgraphs usually represent a structural module rather than a complete RNA conformation. The classic DP scheme for 2D structure energy-minimization can thus be supplemented by additional constructs, enabling the consideration of pseudoknots. Towards that, one needs to access $\text{MFE}_{\text{PK}}(i, j)$, the MFE achieved over a region $[i, j]$ by a conformation compatible with one of the input fat graphs. In other words, one needs an ability to prescribe the span, say $[i, j]$, of the fatgraph occurrence, *i.e.* the values of the extremal anchors, while initiating the dynamic programming.

To ensure this possibility, one simply needs to connect the first and last positions in the minimal fatgraph completion. Indeed, since each arc of the input graph must be represented, any tree decomposition for the completion will feature a bag B including both first and last position (+ additional anchors $S := \{k_1, k_2, \ldots\}$). Moreover, since a tree decomposition is unordered, B can be arbitrarily used as the root, preceded by a root node restricted to anchors (i,j). This yields the following entry point for the DP of a fatgraph γ:

$$\mathrm{r}_\gamma(i,j) := \min_{i<k_1<k_2<\ldots<j} M_B[i,k_1,k_2,\ldots,j]$$

which can be queried from within a classic DP scheme for the secondary structure.

5.2 Energy models

The extension to more realistic energy models is possible through functions evaluating recursive non-crossing substructure; crossing configuration-specific score contributions; and modifications of the algorithms that fill tables for the clique and diagonal cases. The former enables scoring non-crossing substructure in the Turner model and doesn't require changes beyond our discussion on recursive substructures and performing standard non-crossing free energy minimization. Handling multiple fatgraphs as described by disjunction at the top level enables specific scoring of different crossing configurations.

The latter case concerns the scoring of energy within helix expansions. Firstly, we observe that stacking energy between base pairs of the helix can be accounted for with minimal modification of the helix table recursions and therefore does not change the complexity. For this purpose, one introduces additional 'closed' states of the tables (corresponding to the matrix for closed subsequences in non-crossing free energy minimization). To explicitly score interior loops and bulges, the helix table recursions are extended by a case minimizing over the different loops. Naïvely, this would increase the complexity by a linear factor, which is avoided by bounding the loop size, as common in implemented folding algorithms, or without bounding the size following [25].

5.3 Recursive substructures

Recursive substructures consist of secondary structures/occurrences of fatgraphs that are inserted, both in between and within helices, usually through recursive calls to the (augmented) 2D folding scheme.

To enable the insertion of substructures within an helix requires modifications to the helix clique/diagonal rules that are very similar to the ones enabling support for the Turner energy model. Assuming the presence of a base pair (i,j), An insertion can indeed be performed by delimiting a region $[i,k]$ (resp. $[k,j]$) of arbitrary length, leading to an overall MFE of $\mathrm{MFE}_{\mathrm{SS}}(i,k)+\delta$, where δ is the free-energy contributed by the rest of the helix (possibly accounting for additional terms associated with multiloops).

To allow arbitrary sub-structures to be inserted in the gaps between consecutive helices, one can again modify the minimal helix expansion to distinguish the anchors a,b associated with consecutive helices (instead of merging them into a single anchor in our initial exposition). By connecting a and b, one ensures their simultaneous presence in a tagged bag B, whose DP recurrence is then augmented to include an energy contribution $\mathrm{MFE}_{\mathrm{SS}}(a+1,b-1)$.

5.4 Partition functions and ensemble applications

For ensemble applications of our DP schemes, such as computing the partition function [26] and statistical sampling of the Boltzmann ensemble [12], it is imperative for the DP scheme above to be complete and unambiguous [31]. Fortunately, both properties are already guaranteed by our DP schemes. Indeed, intuitively: the completeness is ensured by the exhaustive investigation of all possible anchor positions, i.e. all possible partitions; the unambiguity is guaranteed by the invariant that assigning a position x to a given anchor (within a transitional or diagonal bag), leads x to be paired within the (half-)helix immediately to its right. Choosing different values for x thus induces different innermost/outermost base pairs for the associated helix, leading to disjoint sets of structures.

From this property, we conclude that the partition function for a fatgraph (or several, possibly recursively and/or within a realistic energy model) can be obtained by simply replacing the $(\min, +, \Delta G)$ terms into $(\sum, \times, e^{\beta\Delta G})$, with $\beta = RT$ being the Boltzmann constant multiplied by some absolute temperature.

6 (Re-)Designing algorithms for specific pseudoknot classes

Our pipeline for automated generation of DP folding equations given a fatgraph has been implemented using `Python` and `Snakemake` [28]. The implementation is freely available at:

`https://gitlab.inria.fr/bmarchan/auto-dp`

Since the algorithms in [33] have been described in terms of a finite number of fatgraphs (called irreducible shadows in the paper), one can directly apply our method to obtain an efficient algorithm that covers the same class as `gfold`, namely **1-structures** that are recursive expansions of the four fatgraphs of genus 1 corresponding to simple PK 'H' `([)]`, kissing hairpin 'K' `([)(])`, three-knot 'L' `({[)}]` and 'M' `([{)(]})` (here, represented in *dot-bracket notation*, i.e. corresponding opening and closing brackets correspond to arcs). The maximum complexity of $O(n^6)$ of the four fatgraphs (see Table 1) implies that the automatically derived algorithm covers the class of 1-structures in $O(n^6)$ time—the same complexity as hand-crafted `gfold`. Note that [33] used declarative methods in their algorithm design only to the point of generating grammar rules, which without further optimization yield $O(n^{18})$ (after applying algebraic dynamic programming; ADP [37]). In contrast, our method obtains the optimal complexity in fully automatic fashion. Beyond this re-design of `gfold`, remarkably our method is equally prepared to automatically design a DP algorithm with optimized efficiency for **2-structures**, which are based on all genus 2 fatgraphs. This is remarkable, since the implementation of a practical algorithm has been considered infeasible [33] due to the large number of genus 2 shadows (namely, there are 3472 shadows/fatgraphs), whose grammar rules would have to be optimized by hand. In contrast, due to full automation, our method directly handles even the large number of fatgraphs of genus 2 and yields an efficient, complexity optimized, DP scheme.

Recall that we cover all other pseudoknot classes that are recursive expansions of a finite number of fatgraphs (in the same way as we cover the design of prediction algorithms for 1- and 2-structures). In this way, among the previously existing DP algorithms, we cover the class of **Dirks&Pierce** (D&P) [14], simply by specifying the H-type as single input fatgraph. Consequently, we automatically re-design the D&P algorithm in the same complexity of $O(n^5)$. Even more interestingly, we can design algorithms covering specific (sets of) crossing configurations. This results in an infinite class of efficient algorithms that have not been designed before. Again the complexity of such algorithms is dominated by the most complex fatgraph; where results for interesting ones are given in Table 1. Most remarkably, we design an algorithm optimizing over recursive expansions of kissing hairpins in $O(n^4)$, whereas CCJ [10, 21], which was specifically designed to cover kissing hairpins, requires $O(n^5)$.

A special case, which further showcases the flexibility, is the extension of existing classes by specific crossing configurations. For example, extending D&P by kissing hairpin covers a much larger class while staying in the same complexity. Extending 1-structures by 5-chain yields a new algorithm with a complexity below of 2-structures (namely only $O(n^7)$ instead of

name	fatgraph	treewidth	complexity of folding
H-type	([)]	4	$O(n^5)$
kissing hairpins	([)(])	4	$O(n^4)$ (*)
"L" [33]	([{)]}	5	$O(n^6)$
"M" [33]	([{)(]})	5	$O(n^6)$
4-clique	([{<)]}>	5	$O(n^6)$
5-clique	([{<A)]}>a	5	$O(n^6)$
5-chain	({[)(][)}]	6	$O(n^7)$

Table 1 Table listing pseudoknot classes, corresponding treewidth and resulting complexity of the folding algorithm. In all cases except the one denoted by (*), the complexity of folding is equal to $O(n^{tw+1})$. For the kissing hairpins case, we are in the specific case where the most complex routine is the alignment of a "clique case" helix, which is done in $O(n^4)$ despite a treewidth of 4. These examples are detailed in the Appendix, Figure 9. The DP equations for each of these examples have been automatically generated by a `Python` implementation of our pipeline, freely available at `https://gitlab.inria.fr/bmarchan/auto-dp`.

$O(n^8)$ [33]). The complexity of 5-chain is remarkably low, when considering that previously described algorithms covering this configuration take $O(n^8)$ (e.g. `gfold`'s generalization to 2-structures and a hypothetical blow-up of the Rivas and Eddy algorithm [39] to 6-dimensional instead of 4-dimensional DP matrix elements—both of which have never been implemented).

7 Conclusions and discussion

In this work, we provide an algorithm that takes a family of fatgraphs, i.e. pseudoknotted structures, and returns DP equations that efficiently predict arc annotations minimizing the free energy. The DP equations are automatically generated based on an expansion of the fatgraph, designed to capture helices of arbitrary length. The DP tables in the equations use a number of indices smaller than or equal to the treewidth of the minimal expansion. This very general framework recovers the complexity of prior, hand-crafted algorithms, and lays the foundation for a purely declarative approach to RNA folding with pseudoknots.

In addition to the extensions described in Section 5, this work suggests perspectives that will be explored in future work. Indeed, the choice of an optimal decomposition/DP scheme for the input fatgraph can be seen as the automated design of an optimal table strategy in the context of algebraic dynamic programming [32, 4, 37]. This would enable extensions to multiple context free grammars or tree grammars when describing the problem in the ADP framework.

Our automated design of pseudoknot folding algorithms could naturally be extended to RNA–RNA interactions, since the joint conformation of two interacting RNA sequences can be seen as a pseudoknot when concatenating the two structures [13]. More ambitiously, categories of pseudoknots inducing an infinite family of fatgraphs, *e.g.* as covered by the seminal Rivas & Eddy algorithm [39], could be captured by allowing the introduction of recursive gapped structures in prescribed parts of the fatgraph. This could be addressed by adding cliques to the minimal completion graph would ensure the availability of the relevant anchors in some bags of the tree decomposition, allowing to score such, non-contiguous, recursive substructures.

Another avenue for future research includes a proof of optimality, in term of polynomial complexity, for the produced DP algorithms. Of course, it would be far too ambitious (and erroneous) to expect our DP schemes to be optimal within general computational models. However, it may be possible to prove optimality within a clearly-defined subset of standard implementations of a subset of DP schemes, *e.g.* by contradiction since the existence of a better algorithm would imply the existence of a tree decomposition having smaller width.

References

1 Tatsuya Akutsu. Dynamic programming algorithms for RNA secondary structure prediction with pseudoknots. *Discrete Applied Mathematics*, 104(1-3):45–62, 2000.

2 Can Alkan, Emre Karakoç, Joseph H. Nadeau, S. Cenk Sahinalp, and Kaizhong Zhang. RNA–RNA Interaction Prediction and Antisense RNA Target Search. *Journal of Computational Biology*, 13(2):267–282, 2006. `doi:10.1089/cmb.2006.13.267`.

3 Stefan Arnborg, Derek G Corneil, and Andrzej Proskurowski. Complexity of finding embeddings in ak-tree. *SIAM Journal on Algebraic Discrete Methods*, 8(2):277–284, 1987.

4 Sarah J Berkemer, Christian Höner zu Siederdissen, and Peter F Stadler. Algebraic dynamic programming on trees. *Algorithms*, 10(4):135, 2017.

5 Hans L Bodlaender. A linear-time algorithm for finding tree-decompositions of small treewidth. *SIAM Journal on computing*, 25(6):1305–1317, 1996.

6 Hans L Bodlaender and Arie MCA Koster. Safe separators for treewidth. *Discrete Mathematics*, 306(3):337–350, 2006.

7 Hans L Bodlaender and Arie MCA Koster. Combinatorial optimization on graphs of bounded treewidth. *The Computer Journal*, 51(3):255–269, 2008.

8 Hans L Bodlaender and Arie MCA Koster. Treewidth computations i. upper bounds. *Information and Computation*, 208(3):259–275, 2010.

9 Song Cao and Shi-Jie Chen. Predicting RNA pseudoknot folding thermodynamics. *Nucleic Acids Research*, 34(9):2634–2652, 01 2006. `doi:10.1093/nar/gkl346`.

10 Ho-Lin Chen, Anne Condon, and Hosna Jabbari. An O(n^5) algorithm for MFE prediction of kissing hairpins and 4-chains in nucleic acids. *Journal of Computational Biology*, 16(6):803–815, 2009.

11 Marek Cygan, Fedor V Fomin, Łukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michał Pilipczuk, and Saket Saurabh. *Parameterized algorithms*, volume 1. Springer, 2015.

12 Ye Ding and Charles E. Lawrence. A statistical sampling algorithm for RNA secondary structure prediction. *Nucleic Acids Research*, 31(24):7280–7301, 12 2003. `doi:10.1093/nar/gkg938`.

13 Robert M Dirks, Justin S Bois, Joseph M Schaeffer, Erik Winfree, and Niles A Pierce. Thermodynamic analysis of interacting nucleic acid strands. *SIAM review*, 49(1):65–88, 2007.

14 Robert M Dirks and Niles A Pierce. A partition function algorithm for nucleic acid secondary structure including pseudoknots. *Journal of computational chemistry*, 24(13):1664–1677, 2003.

15 Chuong B Do, Daniel A Woods, and Serafim Batzoglou. CONTRAfold: RNA secondary structure prediction without physics-based models. *Bioinformatics*, 22(14):e90–e98, 2006.

16 Mark E. Fornace, Nicholas J. Porubsky, and Niles A. Pierce. A Unified Dynamic Programming Framework for the Analysis of Interacting Nucleic Acid Strands: Enhanced Models, Scalability, and Speed. *ACS Synthetic Biology*, 9(10):2665–2678, 2020. PMID: 32910644. `doi:10.1021/acssynbio.9b00523`.

17 Robert Giegerich, Björn Voß, and Marc Rehmsmeier. Abstract shapes of rna. *Nucleic acids research*, 32(16):4843–4851, 2004.

18 Vibhav Gogate and Rina Dechter. A complete anytime algorithm for treewidth. *arXiv preprint arXiv:1207.4109*, 2012.

19 Fenix Huang, Christian Reidys, and Reza Rezazadegan. Fatgraph models of RNA structure. *Computational and Mathematical Biophysics*, 5(1):1–20, 2017.

20 Hosna Jabbari and Anne Condon. A fast and robust iterative algorithm for prediction of RNA pseudoknotted secondary structures. *BMC bioinformatics*, 15(1):1–17, 2014.

21 Hosna Jabbari, Ian Wark, Carlo Montemagno, and Sebastian Will. Knotty: efficient and accurate prediction of complex RNA pseudoknot structures. *Bioinformatics*, 34(22):3849–3856, 2018.

22 Martin Loebl and Iain Moffatt. The chromatic polynomial of fatgraphs and its categorification. *Advances in Mathematics*, 217(4):1558–1587, 2008.

23 R Lorenz, SH Bernhart, C Höner Zu Siederdissen, H Tafer, C Flamm, PF Stadler, and IL Hofacker. ViennaRNA Package 2.0. vol. 6. *Algorithms Mol. Biol*, page 26, 2011.

24 László Lovász. Graph minor theory. *Bulletin of the American Mathematical Society*, 43(1):75–86, 2006.

25 R. B. Lyngsø, M. Zuker, and C. N. Pedersen. Fast evaluation of internal loops in RNA secondary structure prediction. *Bioinformatics (Oxford, England)*, 15(6):440–445, June 1999. `doi:10.1093/bioinformatics/15.6.440`.

26 J. S. McCaskill. The equilibrium partition function and base pair binding probabilities for rna secondary structure. *Biopolymers*, 29(6-7):1105–1119, 1990. `doi:https://doi.org/10.1002/bip.360290621`.

27 Mathias Möhl, Sebastian Will, and Rolf Backofen. Lifting prediction to alignment of RNA pseudoknots. *Journal of Computational Biology*, 17(3):429–442, 2010.

28 Felix Mölder, Kim Philipp Jablonski, Brice Letcher, Michael B Hall, Christopher H Tomkins-Tinch, Vanessa Sochat, Jan Forster, Soohyun Lee, Sven O Twardziok, Alexander Kanitz, et al. Sustainable data analysis with snakemake. *F1000Research*, 10, 2021.

29 Ruth Nussinov and Ann B Jacobson. Fast algorithm for predicting the secondary structure of single-stranded rna. *Proceedings of the National Academy of Sciences*, 77(11):6309–6313, 1980.

30 Robert Clark Penner, Michael Knudsen, Carsten Wiuf, and Jørgen Ellegaard Andersen. Fatgraph models of proteins. *Communications on Pure and Applied Mathematics*, 63(10):1249–1297, 2010.

31 Yann Ponty and Cédric Saule. A combinatorial framework for designing (pseudoknotted) RNA algorithms. In Teresa M. Przytycka and Marie-France Sagot, editors, *Algorithms in Bioinformatics*, pages 250–269, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

32 Michela Quadrini, Luca Tesei, and Emanuela Merelli. An algebraic language for RNA pseudoknots comparison. *BMC bioinformatics*, 20(4):1–18, 2019.

33 Christian M Reidys, Fenix WD Huang, Jørgen E Andersen, Robert C Penner, Peter F Stadler, and Markus E Nebel. Topology and prediction of RNA pseudoknots. *Bioinformatics*, 27(8):1076–1085, 2011.

34 Christian M Reidys and Rita R Wang. Shapes of RNA pseudoknot structures. *Journal of Computational Biology*, 17(11):1575–1590, 2010.

35 Jihong Ren, Baharak Rastegari, Anne Condon, and Holger H Hoos. HotKnots: heuristic prediction of RNA secondary structures including pseudoknots. *Rna*, 11(10):1494–1504, 2005.

36 Jessica S Reuter and David H Mathews. RNAstructure: software for rna secondary structure prediction and analysis. *BMC bioinformatics*, 11(1):1–9, 2010.

37 Maik Riechert, Christian Höner zu Siederdissen, and Peter F. Stadler. Algebraic dynamic programming for multiple context-free grammars. *Theoretical Computer Science*, 639:91–109, August 2016. `doi:10.1016/j.tcs.2016.05.032`.

38 Philippe Rinaudo, Yann Ponty, Dominique Barth, and Alain Denise. Tree decomposition and parameterized algorithms for RNA structure-sequence alignment including tertiary interactions and pseudoknots. In *International Workshop on Algorithms in Bioinformatics*, pages 149–164. Springer, 2012.

39 Elena Rivas and Sean R Eddy. A dynamic programming algorithm for RNA structure prediction including pseudoknots. *Journal of molecular biology*, 285(5):2053–2068, 1999.

40 Kengo Sato, Manato Akiyama, and Yasubumi Sakakibara. RNA secondary structure prediction using deep learning with thermodynamic integration. *Nature communications*, 12(1):1–9, 2021.

41 Kengo Sato, Yuki Kato, Michiaki Hamada, Tatsuya Akutsu, and Kiyoshi Asai. IPknot: fast and accurate prediction of RNA secondary structures with pseudoknots using integer programming. *Bioinformatics*, 27(13):i85–i93, 2011.

42 Céline Scornavacca and Mathias Weller. Treewidth-based algorithms for the small parsimony problem on networks. In *WABI*, volume 201 of *LIPIcs*, pages 6:1–6:21. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021.

43 Hisao Tamaki. Positive-instance driven dynamic programming for treewidth. *Journal of Combinatorial Optimization*, 37(4):1283–1311, 2019.

44 Edwin Ten Dam, Kees Pleij, and David Draper. Structural and functional aspects of RNA pseudoknots. *Biochemistry*, 31(47):11665–11676, 1992.

45 Hua-Ting Yao, Jérôme Waldispühl, Yann Ponty, and Sebastian Will. Taming Disruptive Base Pairs to Reconcile Positive and Negative Structural Design of RNA. In *RECOMB 2021-25th international conference on research in computational molecular biology*, 2021.

46 Shay Zakov, Yoav Goldberg, Michael Elhadad, and Michal Ziv-Ukelson. Rich parameterization improves RNA structure prediction. *Journal of Computational Biology*, 18(11):1525–1542, 2011.

47 Michael Zuker. Mfold web server for nucleic acid folding and hybridization prediction. *Nucleic acids research*, 31(13):3406–3415, 2003.

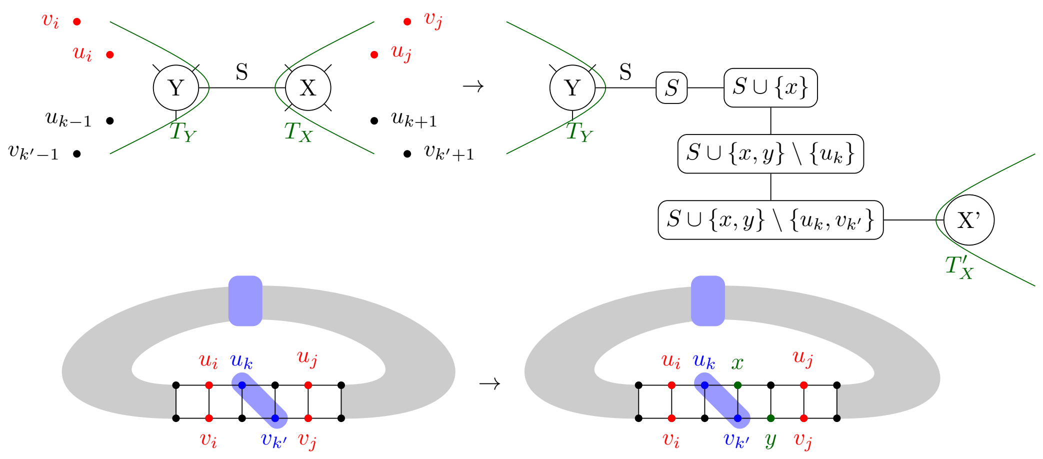


Figure 7 Representation of the local rewriting of a tree decomposition next to a separator S separating to base pairs (u_i, v_i) and (u_j, v_j), in order to extend a helix by one unit, through the introduction of new vertices x and y.

A Width of a helix closed by a clique

Let us denote by H_l^* the graph corresponding to a helix of length l, with the extremities connected as a clique. This graph appears when considering the possible safety (see Proposition 6) of the extremities as a separator of the graph. We show the following result:

▶ **Lemma 14.** *For $l = 2$, $tw(H_l^*) = 3$, while for $l \geq 3$, $tw(H_l^*) = 4$.*

Proof. For $l = 2$, H_l^* is simply the clique on 4 vertices, and which has a width of 3. For $l \geq 3$, a clique on 5 vertices can be obtained as a minor by contracting the internal part of the helix to one vertex, which ends up being connected to all 4 extremities, which already form a clique. Therefore, $tw(H_l^*) \geq 4$. To obtain the equality, we recursively build a tree decomposition of width ≤ 4, starting with $l = 2$ which we already described. Given a tree decomposition of width ≤ 4 for H_l^*, there has to be a bag X containing all 4 extremities $\{u_1, v_1, u_l, v_l\}$ (see Figure 3(b)). We introduce two new bags: $X' = \{u_1, v_1, u_l, v_l, v_{l+1}\}$ introducing a new vertex v_{l+1}, and $X'' = \{u_1, v_1, u_l, v_{l+1}, u_{l+1}\}$ introducing u_{l+1}. We connect X' to X and X'' to X'. By doing so, we respect the subtree connectivity property for all involved vertices, and build a tree decomposition capable of representing H_{l+1}^*. ◀

B Helix extension close to a separator

Figure 7 shows how, once we have found a separator, associated to an edge of the tree decomposition, separating $\{u_i, v_i\}$ from $\{u_j, v_j\}$ with $i < j$, we can insert new vertices in the helix, extending it while preserving the treewidth. This is used in the proof of Theorem 9, in what corresponds in Section 4 to the "diagonal" case.

C Detailed examples

Figure 8 shows a canonical tree decomposition for the minimal length-5 expansion, shown in the upper half of the figure, for the fatgraph showed in Figure 1. This tree decomposition is optimal, and was computed with [43], a solver that empirically works quite fast on RNA graphs.

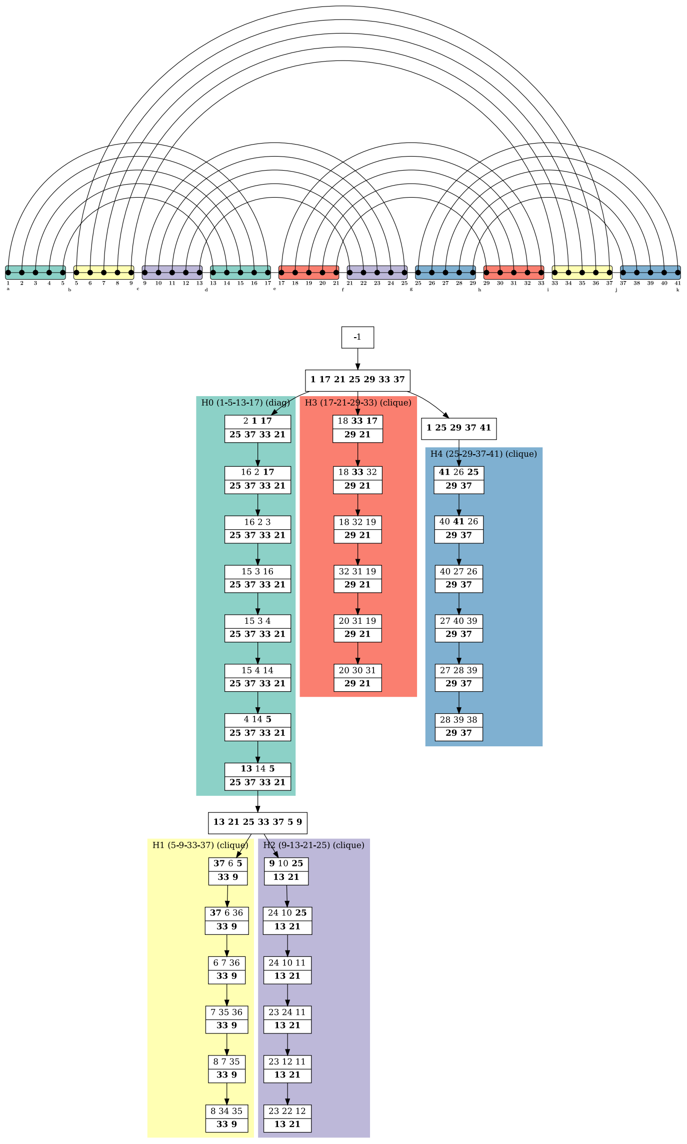
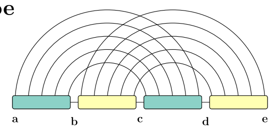


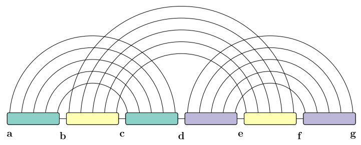
Figure 8 Canonical tree decomposition of the fatgraph given in Figure 1. White boxes represent the bags of the tree decomposition. Number in the bags correspond to the indices of the helices in the fatgraph where number on the bottom are kept while traversing the branch of the decomposition tree. Colored frames indicate the distinct helices (H0 to H4) of the structure.

H-type



$$A = \min_{a,b,c,d,e} \left(C_\boxtimes\,[b, c-1, d, e-1] + C_\boxtimes[a, b-1, c, d-1]\right)$$

kissing hairpins



$$A = \min_{a,d,g} (B\,[a, d|d, g])$$

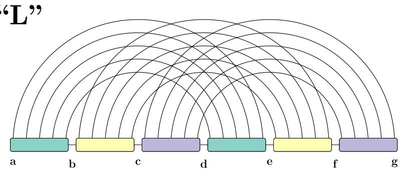
$$B'\,[a, d|d', g] = \min \begin{cases} B'[a, d-1|d', g], & \text{if } d-1, \notin \{a, d', g\} \\ B[a+1, d-1|d', g] + \Delta G(a, d) & \text{if } \{a+1, d-1\} \cap \{d', g\} = \emptyset \end{cases}$$

$$B\,[a, d|d', g] = \min \begin{cases} B[a+1, d|d', g], & \text{if } a+1 \notin \{d, d', g\} \\ B'[a, d-1|d', g], & \text{if } d-1, \notin \{a, d', g\} \\ B[a+1, d-1|d', g] + \Delta G(a, d) & \text{if } \{a+1, d-1\} \cap \{d', g\} = \emptyset, \\ C'[d', g|a, d] \end{cases}$$

$$C'\,[d, g|b, c] = \min \begin{cases} C'[d, g-1|b, c], & \text{if } g-1, \notin \{d, b, c\} \\ C[d+1, g-1|b, c] + \Delta G(d, g) & \text{if } \{d+1, g-1\} \cap \{b, c\} = \emptyset \end{cases}$$

$$C\,[d, g|b, c] = \min \begin{cases} C[d+1, g|b, c], & \text{if } d+1 \notin \{g, b, c\} \\ C'[d, g-1|b, c], & \text{if } g-1, \notin \{d, b, c\} \\ C[d+1, g-1|b, c] + \Delta G(d, g) & \text{if } \{d+1, g-1\} \cap \{b, c\} = \emptyset, \\ C'_\boxtimes[b, c-1, d, g+1-1] \end{cases}$$

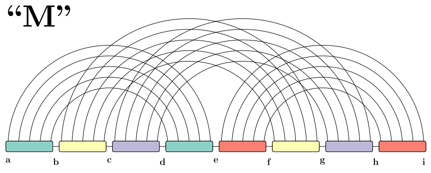
“L”



$$A = \min_{a,c,d,f,g} \left(B\,[a, c, d, f] + C_\boxtimes\,[c, d-1, f, g-1]\right)$$

$$B\,[a, c, d, f] = \min_{b,e} \left(C_\boxtimes\,[b, c-1, e, f-1] + C_\boxtimes\,[a, b-1, d, e-1]\right)$$

“M”

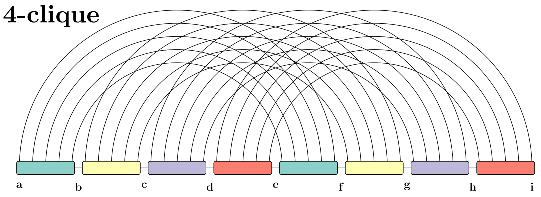


$$A = \min_{a,e,f,h,i} \left(B\,[a, e, f, h] + C_\boxtimes\,[e, f-1, h, i-1]\right)$$

$$B\,[a, e, f, h] = \min_{b,d} \left(C_\boxtimes\,[a, b-1, d, e-1] + C\,[b, d, f, h]\right)$$

$$C\,[b, d, f, h] = \min_{c,g} \left(C_\boxtimes\,[c, d-1, g, h-1] + C_\boxtimes\,[b, c-1, f, g-1]\right)$$

4-clique

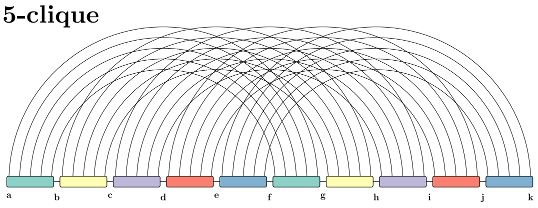


$$A = \min_{a,d,e,h,i} \left(B\,[a, d, e, h] + C_\boxtimes\,[d, e-1, h, i-1]\right)$$

$$B\,[a, d, e, h] = \min_{c,g} \left(C\,[a, c, e, g] + C_\boxtimes\,[c, d-1, g, h-1]\right)$$

$$C\,[a, c, e, g] = \min_{b,f} \left(C_\boxtimes\,[b, c-1, f, g-1] + C_\boxtimes\,[a, b-1, e, f-1]\right)$$

5-clique



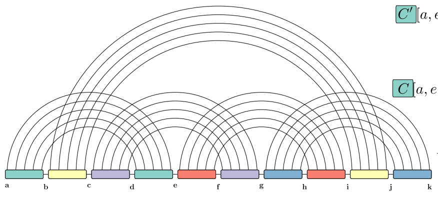
$$A = \min_{a,e,f,j,k} \left(B\,[a, e, f, j] + C_\boxtimes\,[e, f-1, j, k-1]\right)$$

$$B\,[a, e, f, j] = \min_{d,i} \left(C\,[a, d, f, i] + C_\boxtimes\,[d, e-1, i, j-1]\right)$$

$$C\,[a, d, f, i] = \min_{b,g} \left(D\,[b, d, g, i] + C_\boxtimes\,[a, b-1, f, g-1]\right)$$

$$D\,[b, d, g, i] = \min_{c,h} \left(C_\boxtimes\,[c, d-1, h, i-1] + C_\boxtimes\,[b, c-1, g, h-1]\right)$$

5-cycle



$$A = \min_{a,g,h,j,k} \left(B\,[a, g, h, j] + C_\boxtimes\,[g, h-1, j, k-1]\right)$$

$$B\,[a, g, h, j] = \min_{e,f,i} \left(C_\boxtimes\,[e, f-1, h, i-1] + C\,[a, e|f, g, i, j]\right)$$

$$C'[a, e|f, g, i, j] = \min \begin{cases} C'[a, e-1|f, g, i, j], & \text{if } e-1, \notin \{a, f, g, i, j\} \\ C[a+1, e-1|f, g, i, j] + \Delta G(a, e) & \text{if } \{a+1, e-1\} \cap \{f, g, i, j\} = \emptyset \end{cases}$$

$$C[a, e|f, g, i, j] = \min \begin{cases} C[a+1, e|f, g, i, j], & \text{if } a+1 \notin \{e, f, g, i, j\} \\ C'[a, e-1|f, g, i, j], & \text{if } e-1, \notin \{a, f, g, i, j\} \\ C[a+1, e-1|f, g, i, j] + \Delta G(a, e) & \text{if } \{a+1, e-1\} \cap \{f, g, i, j\} = \emptyset, \\ D'[a, e+1, f, g, i, j] \end{cases}$$

$$D\,[b, d, f, g, i, j] = \min_{c} \left(C_\boxtimes\,[c, d-1, f, g-1] + C_\boxtimes\,[b, c-1, i, j-1]\right)$$

Figure 9 Minimal representative expansions and final equations for the examples of Table 1. The equations have been automatically generated, and the pipeline code is freely available at `https://gitlab.inria.fr/bmarchan/auto-dp`. In particular, the optimal tree decompositions were computed by [43].

D Transforming a tree decomposition in its canonical form

Agorithm 2 describes how to obtain a canonical tree decomposition for an RNA structure graph, given any valid tree decomposition as input. Interestingly, it can use a sub-optimal tree decomposition obtained from a polynomial heuristic [7] instead of an exponential solver (although [43] is empirically quite efficient on RNA structure graphs).

The run-time and correctness of Algorithm 2 are stated in Proposition 12.

ALGORITHM 2 Algorithm for re-writing a tree decomposition into a canonical one in which every helix of the input graph is represented in a canonical way

```
Input : A (not necessarily optimal) tree decomposition $\mathcal{T}$ of a minimal expansion of a
        fatgraph $\gamma$.
Output: A tree decomposition of $G$ in canonical form
1  if $width(\mathcal{T}) \leq 3$ then
2      foreach helix $H$ in fatgraph $\gamma$ do
3          if $\exists$ hop-edge represented in $\mathcal{T}$ then
4              use hop-edge to obtain a tree dec. for $G_{\boxtimes}$            //▶ (see Fig. 3(d))
5
6              find a bag $X = \{u_1, v_1, u_l, v_l\}$ as $w(T) \leq 3$
7              replace $X$ with a "diagonal" canonical representation with $S = \emptyset$.
8          else
9              find an edge $(X, Y)$ of $\mathcal{T}$ s.t $X \cap Y$ separates $u_1, v_1$ on the X-side from $u_l, v_l$ on
                 the Y-side
10             $\forall i$, replace $u_i$ with $u_1$ and $v_i$ with $v_1$ in all bags of the $X$-side of $\mathcal{T}$
11             $\forall j$, replace $u_j$ with $u_l$ and $v_j$ with $v_l$ in all bags of the $Y$-side of $\mathcal{T}$
12             Insert between $X$ and $Y$ the "diagonal" canonical representation for $H$, with
                 constant part $S = (X \cap Y) \setminus \{u_k, v_k\}_{i \leq k \leq j}$
13         end
14     end
15 else
16     for helix $H$ in $\gamma$ do
17         if $\exists$ a hop-edge represented in $\mathcal{T}$ then
18             Use the hop-edge to obtain a tree decomposition for $G_{\boxtimes}$
19             find a bag containing all extremities and connect $T_l^{\boxtimes}$ to it
20         else
21             find an edge $(X, Y)$ of $\mathcal{T}$ separating $\{u_1, v_1\}$ and $\{u_l, v_l\}$
22             $\forall i$ replace $u_i$ with $u_1$ and $v_i$ with $v_1$on the $X$-side of $\mathcal{T}$
23             $\forall i$ replace $u_i$ with $u_l$ and $v_i$ with $v_l$ on the $Y$-side of $\mathcal{T}$
24             Insert between $X$ and $Y$ the "diagonal" canonical representation for $H$, with
                 constant part $S = (X \cap Y) \setminus \{u_k, v_k\}_{1 \leq k \leq l}$
25         end
26     end
27 end
```

E Delayed proofs

Proof of Proposition 12. Concerning the run-time, enumerating all pairs $1 \leq i < j \leq l)$ is quadratic in the length of the helix under consideration, which is $O(n)$ in a general graph, while testing a given edge for separation of u_i, v_i and u_j, v_j takes $O(n)$ (through breadth-first search) for each of the $O(n)$ edges of the tree decomposition. As for its correctness: in all cases of the algorithm, representations of edges outside the helices is not affected by the re-writing, while edges inside the edges are accounted for by the canonical representations. ◀

Proof of Proposition 5. To start with, $G_{v \leftarrow u}$ is a minor of G, therefore $tw(G_{v \leftarrow u}) \leq tw(G)$. Then, given an optimal tree decomposition $\mathcal{T}$ for $G_{v \leftarrow u}$, since (v, w) is an edge of this graph, there has to be a bag X containing both vertices. If $tw(G_{v \leftarrow u}) = 1$, then $X = \{v, w\}$

and can be split into two bags $\{v,u\}$ and $\{u,w\}$ to obtain a tree decomposition for G. If $tw(G_{v\leftarrow u}) \geq 2$, then we can simply connect a new bag $\{u,v,w\}$ and connect it to X to obtain again a valid tree decomposition for G of the same width. Therefore $tw(G) \leq tw(G_{v\leftarrow u})$ and we have the equality. ◀

Proof of Theorem 9. Let us distinguish two cases depending on the treewidth of G. For both of them, we consider an optimal tree decomposition $\mathcal{T}$ of G and show how to modify it into a valid tree decomposition for the extended version of G:

- if $tw(G) \leq 3$ then there has to be a pair i,j $(i \leq j)$ of indices $\in [1,l]$ such that $|i-j| > 1$ and neither u_i, v_i or u_j, v_j are present together in one bag. Indeed, if $\forall i,j \in [1,l]$ there was such an "hop edge" represented, then contracting u_k, v_k together $\forall k$ would yield a clique on 5 vertices, which is forbidden if $tw(G) \leq 3$. Given such a pair i,j of indices, there has to be an edge (X,Y) of the tree decomposition that separates all occurrences of u_i, v_i from all occurrences of u_j, v_j. Let us denote $S = X \cap Y$ the separator associated to that edge. By Proposition 7, S can be assumed to be inclusion minimal, and therefore to contain exactly 2 vertices u_k and $v_{k'}$ such that $|k-k'| \leq 1$ and $i \leq k,k' \leq j$. Such a separator is depicted on Figure 3(c), as well as on Figure 7. On this latter Figure, we also depict the re-writing we perform: we introduce two new vertices x and y to the X-side of the separator, as well as intermediary bags between Y and X that will gradually transform u_k, v'_k into x and y. To be specific, we introduce S as a bag between X and Y, and connect it to X through the series of bags $S \cup \{x\}$, $S \cup \{x,y\} \setminus \{u_k\}$, $S \cup \{x,y\} \setminus \{u_k, v'_k\}$ in the case (w.l.o.g) that $k \leq k'$. In addition, all occurences of u_k in X and beyond in the subtree rooted at X and directed away from S are replaced with x and those of v'_k with y. Since $|S| \leq tw(G)$, such a re-writing does not increase the treewidth, while representing all necessary edges for an extension of the helix by one level.
- if $tw(G) \geq 4$, then we consider two sub-cases depending on whether $\mathcal{T}$ represents any "hop-edge" as depicted on Figure 3(d), i.e. an edge between u_k and v_l or v_k and u_l for $|k-l| > 1$. If any such edge is represented (i.e. there exists a bag containing both end-points), then by contracting the parts depicted in green on Figure 3 (d) to the extremity they contain (i.e replacing all occurrences of these vertices in the tree decomposition with their corresponding extremity), we obtain a valid tree decomposition for $G_{\boxtimes}$ of width $\leq tw(G)$. By the inequality of Proposition 8, we get that $tw(G) = \max(4, tw(G_{\boxtimes}))$, and the extremities of the helix are a safe separator. There exists therefor an optimal tree decomposition $\mathcal{T}'$ of G which contains S as a bag, separating the helix from the rest of the graph. By Lemma 14, replacing the sub-tree-decomposition of $\mathcal{T}'$ corresponding to the helix with a tree decomposition for a helix longer by 1 unit does not change the width of this sub-tree-decomposition. If there is no such "hop-edge", then there is an edge (X,Y) in the tree decomposition that separates (u_1, v_1) from (u_l, v_l), and to which we can apply the same re-writing as in the case of $tw(G) \leq 3$.

◀