



**HAL**  
open science

# Automated design of dynamic programming schemes for RNA folding with pseudoknots

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

► **To cite this version:**

Bertrand Marchand, Sebastian Will, Sarah Berkemer, Laurent Bulteau, Yann Ponty. Automated design of dynamic programming schemes for RNA folding with pseudoknots. WABI 2022 - 22nd Workshop on Algorithms in Bioinformatics, Sep 2022, Potsdam, Germany. hal-03676377v2

**HAL Id: hal-03676377**

**<https://inria.hal.science/hal-03676377v2>**

Submitted on 11 Aug 2022

**HAL** is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.



Distributed under a Creative Commons Attribution 4.0 International License

# 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<sup>1</sup>  

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>

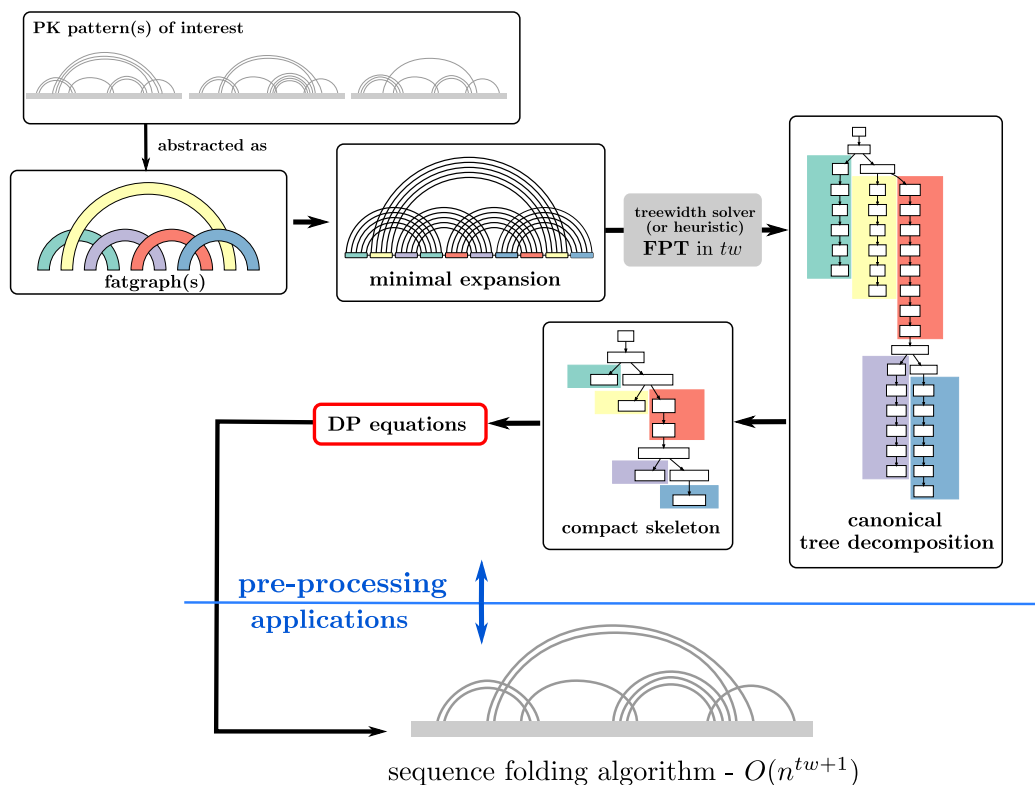
**Supplementary Material** (implementation) <https://gitlab.inria.fr/bmarchan/auto-dp>

**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 PaRNAAssus project (ANR-19-CE45-0023; I 4520-N) supported by the ANR/FWF agencies.

---

<sup>1</sup> To whom correspondence should be addressed





■ **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.

## 41 1 Introduction

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

53 However, the most frequently used algorithms (including all of the above ones) optimize  
 54 solely over pseudoknot-free structures [44], which do not contain crossing base pairs. Although  
 55 pseudoknots appear in many RNA secondary structures, they have been omitted by initial

56 prediction algorithms due to their computational complexity [1], and the difficulty to score  
 57 individual conformations [9]. Nevertheless, many algorithms have been proposed to predict  
 58 at least certain pseudoknots. These methods are either based on exact DP algorithms such  
 59 as `pknots-RE` [39], `NUPACK` [14], `gfold` [33], `Knotty` [21] or they use heuristics that don't  
 60 guarantee exact solutions, e.g., `HotKnots` [35], `IPknot` [41, 40], `Hfold` [20].

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

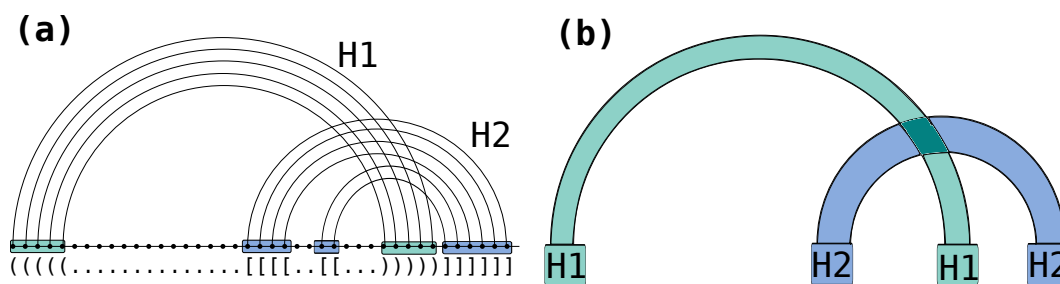
74 In this work, we describe classes of pseudoknotted structures as fatgraphs [19, 33, 22, 30],  
 75 an abstraction of RNA conformations related to RNA shapes [17] or shadows [34, 33]. We  
 76 formalize the principles underlying the design of DP folding algorithms including pseudoknots,  
 77 and,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

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

## 90 **2** Definitions and main result

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

99 One of our central concerns is the crossing configuration of arcs in a diagram. We define  
 100 two arcs  $(i, j)$  and  $(i', j')$  in a diagram as *crossing* iff  $i < i' < j < j'$  or  $i' < i < j' < j$ .  
 101 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.

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

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

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

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

128 ► **Definition 1** ((Recursive) fatgraph MFE folding problem).

129 **Input:** Finite collection of fatgraphs  $\gamma_1, \dots, \gamma_p$ , sequence  $S$

130 **Output:** Minimum Free Energy (MFE) arc-annotation for  $S$  according to free-energy model  
 131  $\mathcal{E}$ , restricting the search to recursive expansions of the input fatgraphs.

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

134 ► **Definition 2** (Fatgraph folding algorithm design problem).

135 **Input:** Finite collection of fatgraphs  $\gamma_1, \dots, \gamma_p$ , sequence  $S$

---

■ **ALGORITHM 1** Pseudocode for the recursive fatgraph folding problem.

**Input** : Finite number of fatgraphs  $\gamma_1, \dots, \gamma_p$ , 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

```

---

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

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

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

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

### 151 3 Minimal representative expansion of a fatgraph

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

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

#### 161 3.1 Treewidth and tree decompositions

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

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

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

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

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

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

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

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

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

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

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

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

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

## 204 3.2 Helices of length 5 are sufficient to obtain generalizable tree 205 decompositions

206 Given an RNA graph (with one vertex per nucleotide and one edge per base pair and backbone  
 207 link, see Figure 3(a)), we call *perfect helix* a set of directly nested base pairs, resulting in

208 the subgraph depicted on Figure 3(b). We call the number of nested base pairs its *length*,  
 209 and denote it with  $l$ . With a slight abuse of language, we call such a subgraph a *helix*, even  
 210 for general graphs. Our main structural result is to show that the treewidth of a graph  $G$   
 211 does not increase when extending a helix past a length of 5. Its proof relies on the following  
 212 inequality, involving the graphs  $G_{\boxtimes}$  and  $G_{\square}$ , obtained from  $G$  by replacing a helix  $H$  with  
 213 either  $\boxtimes$  or  $\square$ , (see Figure 3(c)).

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

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

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

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

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

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

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

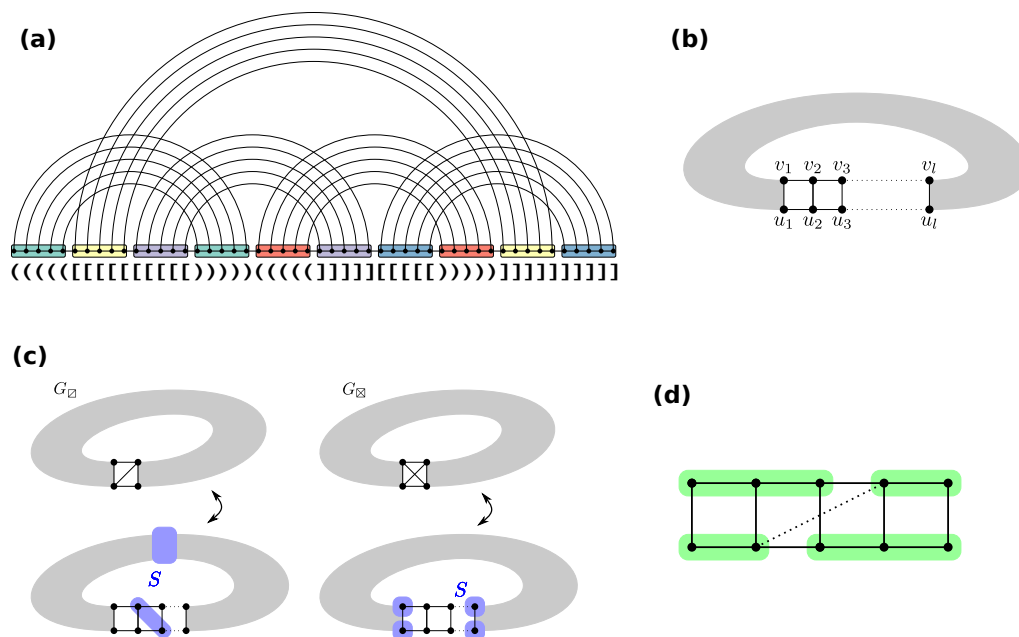
235 Since bulges in a helix only consist of vertices of degree exactly 2, combining Proposition 5  
 236 with Theorem 9 implies that the treewidth of any expansion of a given fatgraph is always  
 237 smaller than or equal to the treewidth of a minimal expansion where all bands are helices of  
 238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

241 ► **Definition 10** (Minimal representative expansion of a fatgraph). *Given a fatgraph  $\gamma$ , its  
 242 minimal representative expansion consists of:*

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

245 Such a minimal representative expansion is illustrated in Figure 5(a). For visual clarity,  
 246 gaps have been kept between consecutive helices, but one can see that the corresponding  
 247 extremities have the same labels. Given a fatgraph, this RNA structure graph contains  
 248 all necessary information for formulating DP equations decomposing all RNA structures  
 249 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  $tw(G_{\square})$ . 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.

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

#### 256 **4 Tree decompositions of fatgraph expansions as RNA DP algorithms**

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

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

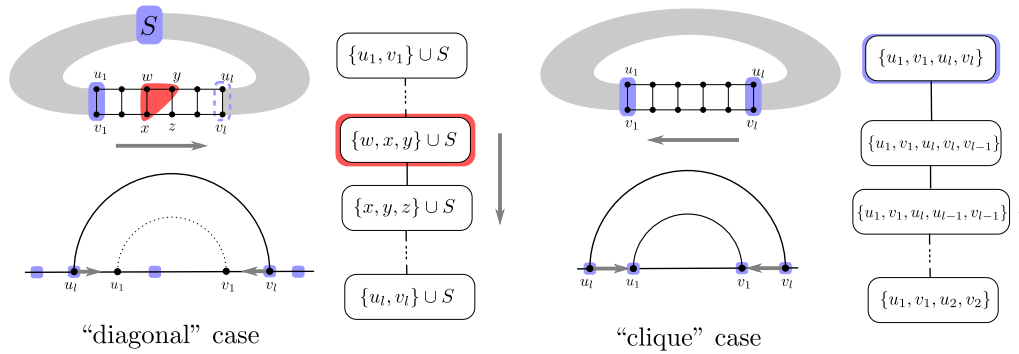


Figure 4 Illustration of the two types of canonical representations for the helices of a graph  $G$

### 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  $\mathcal{T} \setminus e$  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

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

■ **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  $\mathcal{T}$ , 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

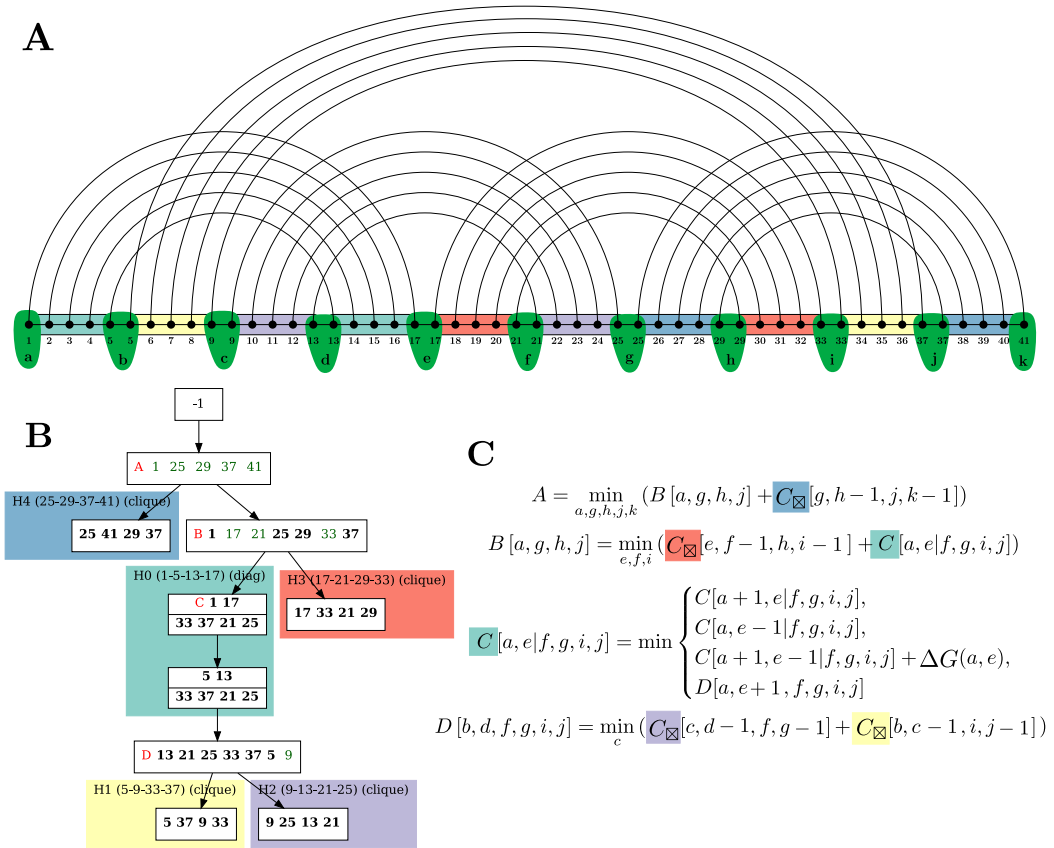
297 ■ All sub-tree-decompositions representing a helix in the “diagonal” case are contracted  
 298 to contain their first and last bags only, denoted as  $S \cup \{u_1, v_1\}$  and  $S \cup \{u_l, v_l\}$  in  
 299 Definition 11.

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

### 301 4.2 Automatic derivation of dynamic programming equations

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

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



■ **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  $(a, b, c, \dots)$  partitions the sequence into a set of disjoint intervals  $[a, b], [b, c], \dots$ , 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  $\geq 1$  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.

### 339 4.2.2 Helix tables 2: “Diagonal” cases

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

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

Namely, let the cell  $D_H[i, j | S]$  (resp.  $D'_H[i, j | S]$ ) 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:

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

and

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

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

### 351 4.2.3 Transitional tables: Non-helix bags

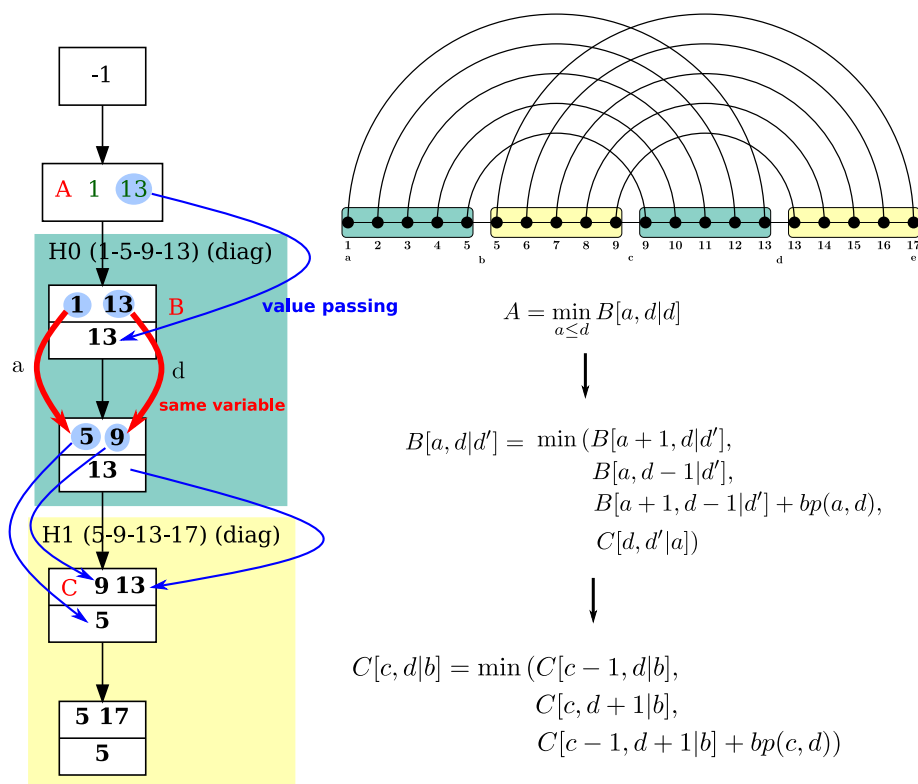
The general case consists of passing the values of relevant variables onward to the diagonal  
 and clique tables, possibly assigning/propagating anchors that appear in the bag for the first  
 time. Let  $I_P$  be the anchors of the parent bag of  $M$  in the tree decomposition, we have:

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

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

## 354 5 Extensions

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



■ **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.

### 362 5.1 Multiple fatgraphs and integration within 2D folding scheme

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

368 However, fatgraphs usually represent a structural module rather than a complete RNA  
 369 conformation. The classic DP scheme for 2D structure energy-minimization can thus be  
 370 supplemented by additional constructs, enabling the consideration of pseudoknots. Towards  
 371 that, one needs to access  $MFE_{PK}(i, j)$ , the MFE achieved over a region  $[i, j]$  by a conformation  
 372 compatible with one of the input fat graphs. In other words, one needs an ability to prescribe  
 373 the span, say  $[i, j]$ , of the fatgraph occurrence, *i.e.* the values of the extremal anchors, while  
 374 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  $S := \{k_1, k_2, \dots\}$ ). 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  $\gamma$ :

$$r_\gamma(i, j) := \min_{i < k_1 < k_2 < \dots < j} M_B[i, k_1, k_2, \dots, j]$$

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

## 376 5.2 Energy models

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

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

## 393 5.3 Recursive substructures

394 Recursive substructures consist of secondary structures/occurrences of fatgraphs that are  
395 inserted, both in between and within helices, usually through recursive calls to the (augmented)  
396 2D folding scheme.

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

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

## 408 5.4 Partition functions and ensemble applications

409 For ensemble applications of our DP schemes, such as computing the partition function [26]  
410 and statistical sampling of the Boltzmann ensemble [12], it is imperative for the DP scheme  
411 above to be complete and unambiguous [31]. Fortunately, both properties are already  
412 guaranteed by our DP schemes. Indeed, intuitively: the completeness is ensured by the  
413 exhaustive investigation of all possible anchor positions, i.e. all possible partitions; the

414 unambiguity is guaranteed by the invariant that assigning a position  $x$  to a given anchor  
 415 (within a transitional or diagonal bag), leads  $x$  to be paired within the (half-)helix immediately  
 416 to its right. Choosing different values for  $x$  thus induces different innermost/outermost base  
 417 pairs for the associated helix, leading to disjoint sets of structures.

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

## 422 **6 (Re-)Designing algorithms for specific pseudoknot classes**

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

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

426 Since the algorithms in [33] have been described in terms of a finite number of fatgraphs  
 427 (called irreducible shadows in the paper), one can directly apply our method to obtain  
 428 an efficient algorithm that covers the same class as `gfold`, namely **1-structures** that are  
 429 recursive expansions of the four fatgraphs of genus 1 corresponding to simple PK 'H' (`[]`),  
 430 kissing hairpin 'K' (`[[]]`), three-knot 'L' (`{[]}`) and 'M' (`[{}]`) (here, represented in  
 431 *dot-bracket notation*, i.e. corresponding opening and closing brackets correspond to arcs).  
 432 The maximum complexity of  $O(n^6)$  of the four fatgraphs (see Table 1) implies that the  
 433 automatically derived algorithm covers the class of 1-structures in  $O(n^6)$  time—the same  
 434 complexity as hand-crafted `gfold`. Note that [33] used declarative methods in their algorithm  
 435 design only to the point of generating grammar rules, which without further optimization  
 436 yield  $O(n^{18})$  (after applying algebraic dynamic programming; ADP [37]). In contrast, our  
 437 method obtains the optimal complexity in fully automatic fashion. Beyond this re-design of  
 438 `gfold`, remarkably our method is equally prepared to automatically design a DP algorithm  
 439 with optimized efficiency for **2-structures**, which are based on all genus 2 fatgraphs. This is  
 440 remarkable, since the implementation of a practical algorithm has been considered infeasible  
 441 [33] due to the large number of genus 2 shadows (namely, there are 3472 shadows/fatgraphs),  
 442 whose grammar rules would have to be optimized by hand. In contrast, due to full automation,  
 443 our method directly handles even the large number of fatgraphs of genus 2 and yields an  
 444 efficient, complexity optimized, DP scheme.

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

456 A special case, which further showcases the flexibility, is the extension of existing classes  
 457 by specific crossing configurations. For example, extending D&P by kissing hairpin covers a  
 458 much larger class while staying in the same complexity. Extending 1-structures by 5-chain  
 459 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>.

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

## 464 7 Conclusions and discussion

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

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

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

487 Another avenue for future research includes a proof of optimality, in term of polynomial  
 488 complexity, for the produced DP algorithms. Of course, it would be far too ambitious (and

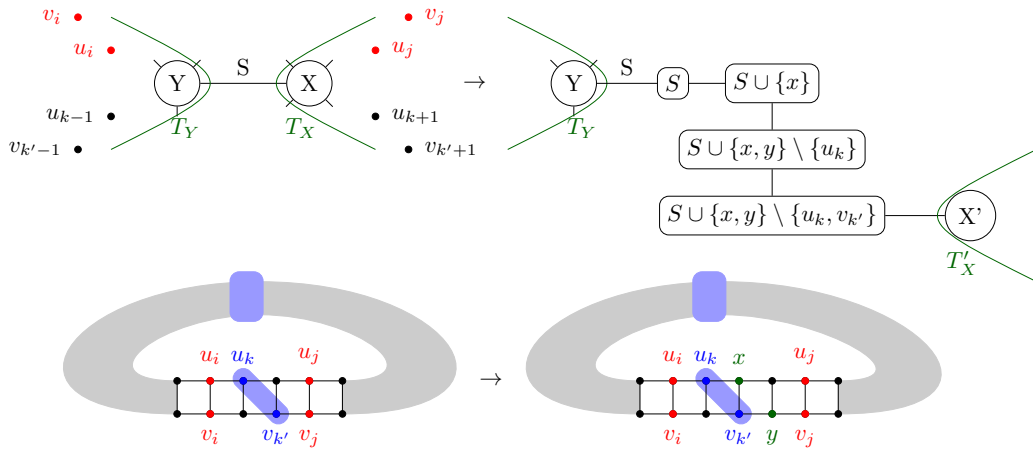
erroneous) to expect our DP schemes to be optimal within general computational models. However, it may be possible to prove optimality within a clearly-defined subset of standard implementations of a subset of DP schemes, *e.g.* by contradiction since the existence of a better algorithm would imply the 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 Siederdisen, 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 Lokshantov, 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.

- 539 21 Hosna Jabbari, Ian Wark, Carlo Montemagno, and Sebastian Will. Knotty: efficient and  
540 accurate prediction of complex RNA pseudoknot structures. *Bioinformatics*, 34(22):3849–3856,  
541 2018.
- 542 22 Martin Loebl and Iain Moffatt. The chromatic polynomial of fatgraphs and its categorification.  
543 *Advances in Mathematics*, 217(4):1558–1587, 2008.
- 544 23 R Lorenz, SH Bernhart, C Höner Zu Siederdisen, H Tafer, C Flamm, PF Stadler, and  
545 IL Hofacker. ViennaRNA Package 2.0. vol. 6. *Algorithms Mol. Biol*, page 26, 2011.
- 546 24 László Lovász. Graph minor theory. *Bulletin of the American Mathematical Society*, 43(1):75–  
547 86, 2006.
- 548 25 R. B. Lyngsø, M. Zuker, and C. N. Pedersen. Fast evaluation of internal loops in RNA  
549 secondary structure prediction. *Bioinformatics (Oxford, England)*, 15(6):440–445, June 1999.  
550 doi:10.1093/bioinformatics/15.6.440.
- 551 26 J. S. McCaskill. The equilibrium partition function and base pair binding probabilities for rna  
552 secondary structure. *Biopolymers*, 29(6-7):1105–1119, 1990. doi:https://doi.org/10.1002/  
553 bip.360290621.
- 554 27 Mathias Möhl, Sebastian Will, and Rolf Backofen. Lifting prediction to alignment of RNA  
555 pseudoknots. *Journal of Computational Biology*, 17(3):429–442, 2010.
- 556 28 Felix Mölder, Kim Philipp Jablonski, Brice Letcher, Michael B Hall, Christopher H Tomkins-  
557 Tinch, Vanessa Sochat, Jan Forster, Soohyun Lee, Sven O Twardziok, Alexander Kanitz, et al.  
558 Sustainable data analysis with snakemake. *F1000Research*, 10, 2021.
- 559 29 Ruth Nussinov and Ann B Jacobson. Fast algorithm for predicting the secondary structure of  
560 single-stranded rna. *Proceedings of the National Academy of Sciences*, 77(11):6309–6313, 1980.
- 561 30 Robert Clark Penner, Michael Knudsen, Carsten Wiuf, and Jørgen Ellegaard Andersen.  
562 Fatgraph models of proteins. *Communications on Pure and Applied Mathematics*, 63(10):1249–  
563 1297, 2010.
- 564 31 Yann Ponty and Cédric Saule. A combinatorial framework for designing (pseudoknotted)  
565 RNA algorithms. In Teresa M. Przytycka and Marie-France Sagot, editors, *Algorithms in*  
566 *Bioinformatics*, pages 250–269, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.
- 567 32 Michela Quadrini, Luca Tesei, and Emanuela Merelli. An algebraic language for RNA  
568 pseudoknots comparison. *BMC bioinformatics*, 20(4):1–18, 2019.
- 569 33 Christian M Reidys, Fenix WD Huang, Jørgen E Andersen, Robert C Penner, Peter F  
570 Stadler, and Markus E Nebel. Topology and prediction of RNA pseudoknots. *Bioinformatics*,  
571 27(8):1076–1085, 2011.
- 572 34 Christian M Reidys and Rita R Wang. Shapes of RNA pseudoknot structures. *Journal of*  
573 *Computational Biology*, 17(11):1575–1590, 2010.
- 574 35 Jihong Ren, Baharak Rastegari, Anne Condon, and Holger H Hoos. HotKnots: heuristic  
575 prediction of RNA secondary structures including pseudoknots. *Rna*, 11(10):1494–1504, 2005.
- 576 36 Jessica S Reuter and David H Mathews. RNAstructure: software for rna secondary structure  
577 prediction and analysis. *BMC bioinformatics*, 11(1):1–9, 2010.
- 578 37 Maik Riechert, Christian Höner zu Siederdisen, and Peter F. Stadler. Algebraic dynamic  
579 programming for multiple context-free grammars. *Theoretical Computer Science*, 639:91–109,  
580 August 2016. doi:10.1016/j.tcs.2016.05.032.
- 581 38 Philippe Rinaudo, Yann Ponty, Dominique Barth, and Alain Denise. Tree decomposition and  
582 parameterized algorithms for RNA structure-sequence alignment including tertiary interactions  
583 and pseudoknots. In *International Workshop on Algorithms in Bioinformatics*, pages 149–164.  
584 Springer, 2012.
- 585 39 Elena Rivas and Sean R Eddy. A dynamic programming algorithm for RNA structure prediction  
586 including pseudoknots. *Journal of molecular biology*, 285(5):2053–2068, 1999.
- 587 40 Kengo Sato, Manato Akiyama, and Yasubumi Sakakibara. RNA secondary structure prediction  
588 using deep learning with thermodynamic integration. *Nature communications*, 12(1):1–9, 2021.

- 589 41 Kengo Sato, Yuki Kato, Michiaki Hamada, Tatsuya Akutsu, and Kiyoshi Asai. IPknot:  
590 fast and accurate prediction of RNA secondary structures with pseudoknots using integer  
591 programming. *Bioinformatics*, 27(13):i85–i93, 2011.
- 592 42 Céline Scornavacca and Mathias Weller. Treewidth-based algorithms for the small parsimony  
593 problem on networks. In *WABI*, volume 201 of *LIPICs*, pages 6:1–6:21. Schloss Dagstuhl -  
594 Leibniz-Zentrum für Informatik, 2021.
- 595 43 Hisao Tamaki. Positive-instance driven dynamic programming for treewidth. *Journal of*  
596 *Combinatorial Optimization*, 37(4):1283–1311, 2019.
- 597 44 Edwin Ten Dam, Kees Pleij, and David Draper. Structural and functional aspects of RNA  
598 pseudoknots. *Biochemistry*, 31(47):11665–11676, 1992.
- 599 45 Hua-Ting Yao, Jérôme Waldispühl, Yann Ponty, and Sebastian Will. Taming Disruptive Base  
600 Pairs to Reconcile Positive and Negative Structural Design of RNA. In *RECOMB 2021-25th*  
601 *international conference on research in computational molecular biology*, 2021.
- 602 46 Shay Zakov, Yoav Goldberg, Michael Elhadad, and Michal Ziv-Ukelson. Rich parameterization  
603 improves RNA structure prediction. *Journal of Computational Biology*, 18(11):1525–1542,  
604 2011.
- 605 47 Michael Zuker. Mfold web server for nucleic acid folding and hybridization prediction. *Nucleic*  
606 *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  $y$ .

607 **A Width of a helix closed by a clique**

608 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  
609 Proposition 6) of the extremities as a separator of the graph. We show the following result:  
610

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

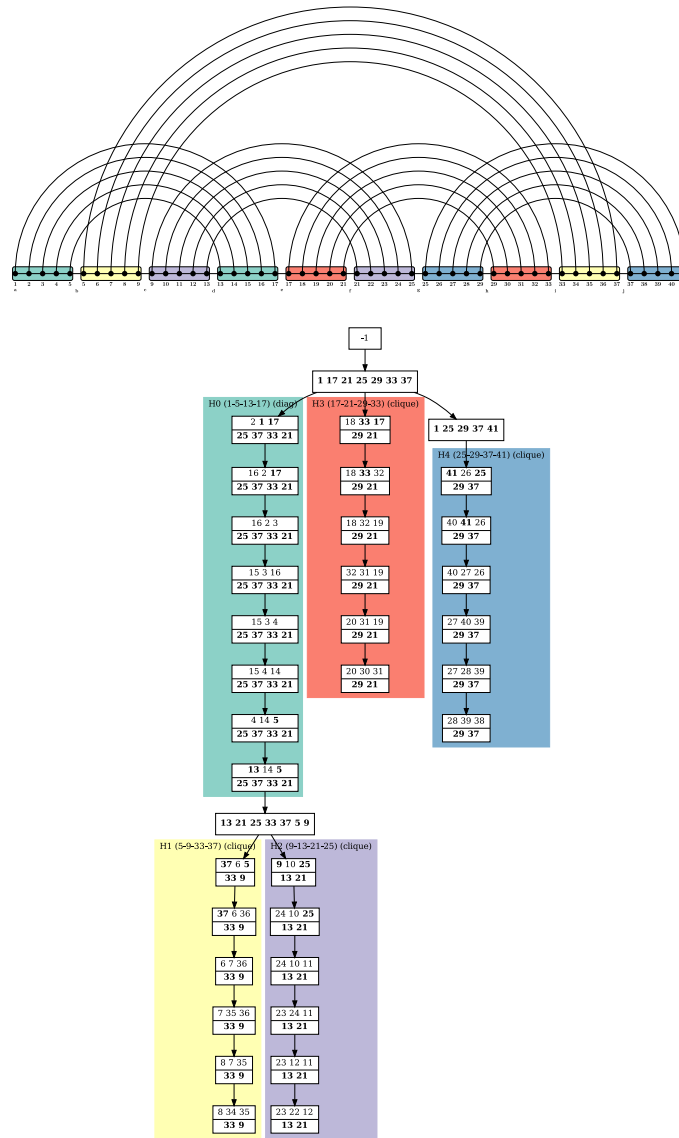
612 **Proof.** For  $l = 2$ ,  $H_l^*$  is simply the clique on 4 vertices, and which has a width of 3. For  $l \geq 3$ ,  
613 a clique on 5 vertices can be obtained as a minor by contracting the internal part of the helix  
614 to one vertex, which ends up being connected to all 4 extremities, which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

622 **B Helix extension close to a separator**

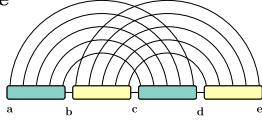
623 Figure 7 shows how, once we have found a separator, associated to an edge of the tree  
624 decomposition, separating  $\{u_i, v_i\}$  from  $\{u_j, v_j\}$  with  $i < j$ , we can insert new vertices in the  
625 helix, extending it while preserving the treewidth. This is used in the proof of Theorem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

627 **C Detailed examples**

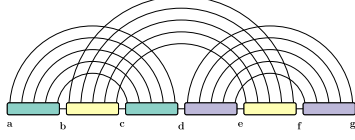
628 Figure 8 shows a canonical tree decomposition for the minimal length-5 expansion, shown in  
629 the upper half of the figure, for the fatgraph showed in Figure 1. This tree decomposition  
630 is optimal, and was computed with [43], a solver that empirically works quite fast on RNA  
631 graphs.



■ **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**


$$A = \min_{a,b,c,d,e} (\mathbb{C}_{\mathbb{R}}[b, c-1, d, e-1] + \mathbb{C}_{\mathbb{R}}[a, b-1, c, d-1])$$

**kissing hairpins**


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

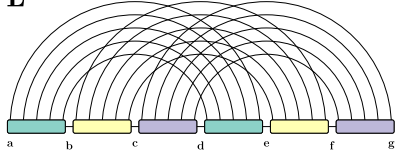
$$\mathbb{B}[a, d|d', g] = \min \begin{cases} B'[a, d-1|d', g], & \text{if } d-1 \notin \{a, d', g\} \\ B[a+1, d-1|d', g] + \Delta G(a, d) & \text{if } \{a+1, d-1\} \cap \{d', g\} = \emptyset \end{cases}$$

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

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

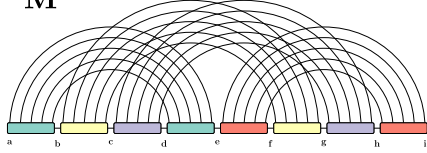
$$\mathbb{C}[d, g|b, c] = \min \begin{cases} C[d+1, g|b, c], & \text{if } d+1 \notin \{g, b, c\} \\ C'[d, g-1|b, c], & \text{if } g-1 \notin \{d, b, c\} \\ C[d+1, g-1|b, c] + \Delta G(d, g) & \text{if } \{d+1, g-1\} \cap \{b, c\} = \emptyset, \end{cases}$$

$$\mathbb{C}_{\mathbb{R}}[b, c-1, d, g+1-1]$$

**“L”**


$$A = \min_{a,c,d,f,g} (B[a, c, d, f] + \mathbb{C}_{\mathbb{R}}[c, d-1, f, g-1])$$

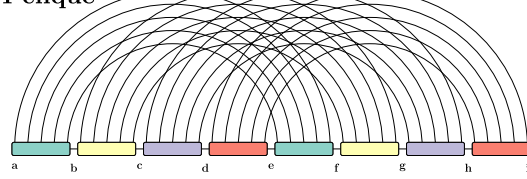
$$B[a, c, d, f] = \min_{b,e} (\mathbb{C}_{\mathbb{R}}[b, c-1, e, f-1] + \mathbb{C}_{\mathbb{R}}[a, b-1, d, e-1])$$

**“M”**


$$A = \min_{a,e,f,h,i} (B[a, e, f, h] + \mathbb{C}_{\mathbb{R}}[e, f-1, h, i-1])$$

$$B[a, e, f, h] = \min_{b,d} (\mathbb{C}_{\mathbb{R}}[a, b-1, d, e-1] + C[b, d, f, h])$$

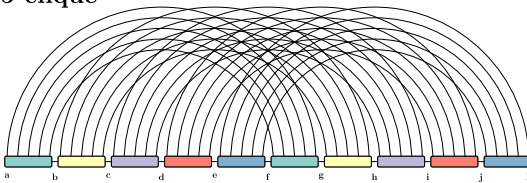
$$C[b, d, f, h] = \min_{c,g} (\mathbb{C}_{\mathbb{R}}[c, d-1, g, h-1] + \mathbb{C}_{\mathbb{R}}[b, c-1, f, g-1])$$

**4-clique**


$$A = \min_{a,d,e,h,i} (B[a, d, e, h] + \mathbb{C}_{\mathbb{R}}[d, e-1, h, i-1])$$

$$B[a, d, e, h] = \min_{c,g} (C[a, c, e, g] + \mathbb{C}_{\mathbb{R}}[c, d-1, g, h-1])$$

$$C[a, c, e, g] = \min_{b,f} (\mathbb{C}_{\mathbb{R}}[b, c-1, f, g-1] + \mathbb{C}_{\mathbb{R}}[a, b-1, e, f-1])$$

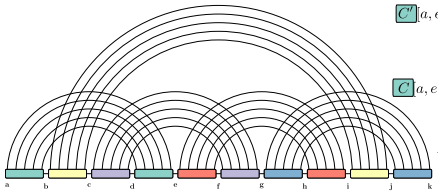
**5-clique**


$$A = \min_{a,e,f,j,k} (B[a, e, f, j] + \mathbb{C}_{\mathbb{R}}[e, f-1, j, k-1])$$

$$B[a, e, f, j] = \min_{d,i} (C[a, d, f, i] + \mathbb{C}_{\mathbb{R}}[d, e-1, i, j-1])$$

$$C[a, d, f, i] = \min_{b,g} (D[b, d, g, i] + \mathbb{C}_{\mathbb{R}}[a, b-1, f, g-1])$$

$$D[b, d, g, i] = \min_{c,h} (\mathbb{C}_{\mathbb{R}}[c, d-1, h, i-1] + \mathbb{C}_{\mathbb{R}}[b, c-1, g, h-1])$$

**5-cycle**


$$A = \min_{a,g,h,j,k} (B[a, g, h, j] + \mathbb{C}_{\mathbb{R}}[g, h-1, j, k-1])$$

$$B[a, g, h, j] = \min_{e,f,i} (\mathbb{C}_{\mathbb{R}}[e, f-1, h, i-1] + C[a, e|f, g, i, j])$$

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

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

$$D[b, d, f, g, i, j] = \min_c (\mathbb{C}_{\mathbb{R}}[c, d-1, f, g-1] + \mathbb{C}_{\mathbb{R}}[b, c-1, i, j-1])$$

■ **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

Algorithm 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
10        the Y-side
11        $\forall i$ , replace  $u_i$  with  $u_1$  and  $v_i$  with  $v_1$  in all bags of the X-side of  $\mathcal{T}$ 
12        $\forall j$ , replace  $u_j$  with  $u_l$  and  $v_j$  with  $v_l$  in all bags of the Y-side of  $\mathcal{T}$ 
13       Insert between  $X$  and  $Y$  the “diagonal” canonical representation for  $H$ , with
14       constant part  $S = (X \cap Y) \setminus \{u_k, v_k\}_{i \leq k \leq j}$ 
15     end
16   end
17 else
18   for helix  $H$  in  $\gamma$  do
19     if  $\exists$  a hop-edge represented in  $\mathcal{T}$  then
20       Use the hop-edge to obtain a tree decomposition for  $G_{\boxtimes}$ 
21       find a bag containing all extremities and connect  $T_l^{\boxtimes}$  to it
22     else
23       find an edge  $(X, Y)$  of  $\mathcal{T}$  separating  $\{u_1, v_1\}$  and  $\{u_l, v_l\}$ 
24        $\forall i$  replace  $u_i$  with  $u_1$  and  $v_i$  with  $v_1$  on the X-side of  $\mathcal{T}$ 
25        $\forall i$  replace  $u_i$  with  $u_l$  and  $v_i$  with  $v_l$  on the Y-side of  $\mathcal{T}$ 
26       Insert between  $X$  and  $Y$  the “diagonal” canonical representation for  $H$ , with
27       constant part  $S = (X \cap Y) \setminus \{u_k, v_k\}_{1 \leq k \leq l}$ 
28     end
29   end
30 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  $X = \{v, w\}$



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

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

- 655 ■ if  $tw(G) \leq 3$  then there has to be a pair  $i, j$  ( $i \leq j$ ) of indices  $\in [1, l]$  such that  $|i - j| > 1$   
 656 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  
 657 was such an “hop edge” represented, then contracting  $u_k, v_k$  together  $\forall k$  would yield a  
 658 clique on 5 vertices, which is forbidden if  $tw(G) \leq 3$ . Given such a pair  $i, j$  of indices,  
 659 there has to be an edge  $(X, Y)$  of the tree decomposition that separates all occurrences  
 660 of  $u_i, v_i$  from all occurrences of  $u_j, v_j$ . Let us denote  $S = X \cap Y$  the separator associated  
 661 to that edge. By Proposition 7,  $S$  can be assumed to be inclusion minimal, and therefore  
 662 to contain exactly 2 vertices  $u_k$  and  $v_{k'}$  such that  $|k - k'| \leq 1$  and  $i \leq k, k' \leq j$ . Such  
 663 a separator is depicted on Figure 3(c), as well as on Figure 7. On this latter Figure,  
 664 we also depict the re-writing we perform: we introduce two new vertices  $x$  and  $y$  to  
 665 the  $X$ -side of the separator, as well as intermediary bags between  $Y$  and  $X$  that will  
 666 gradually transform  $u_k, v_{k'}$  into  $x$  and  $y$ . To be specific, we introduce  $S$  as a bag between  
 667  $X$  and  $Y$ , and connect it to  $X$  through the series of bags  $S \cup \{x\}$ ,  $S \cup \{x, y\} \setminus \{u_k\}$ ,  
 668  $S \cup \{x, y\} \setminus \{u_k, v_{k'}\}$  in the case (w.l.o.g) that  $k \leq k'$ . In addition, all occurrences of  $u_k$  in  
 669  $X$  and beyond in the subtree rooted at  $X$  and directed away from  $S$  are replaced with  
 670  $x$  and those of  $v_{k'}$  with  $y$ . Since  $|S| \leq tw(G)$ , such a re-writing does not increase the  
 671 treewidth, while representing all necessary edges for an extension of the helix by one level.
- 672 ■ if  $tw(G) \geq 4$ , then we consider two sub-cases depending on whether  $\mathcal{T}$  represents any  
 673 “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  
 674  $|k - l| > 1$ . If any such edge is represented (i.e. there exists a bag containing both end-  
 675 points), then by contracting the parts depicted in green on Figure 3 (d) to the extremity  
 676 they contain (i.e replacing all occurrences of these vertices in the tree decomposition with  
 677 their corresponding extremity), we obtain a valid tree decomposition for  $G_{\boxtimes}$  of width  
 678  $\leq tw(G)$ . By the inequality of Proposition 8, we get that  $tw(G) = \max(4, tw(G_{\boxtimes}))$ , and  
 679 the extremities of the helix are a safe separator. There exists therefor an optimal tree  
 680 decomposition  $\mathcal{T}'$  of  $G$  which contains  $S$  as a bag, separating the helix from the rest  
 681 of the graph. By Lemma 14, replacing the sub-tree-decomposition of  $\mathcal{T}'$  corresponding  
 682 to the helix with a tree decomposition for a helix longer by 1 unit does not change the  
 683 width of this sub-tree-decomposition. If there is no such “hop-edge”, then there is an  
 684 edge  $(X, Y)$  in the tree decomposition that separates  $(u_1, v_1)$  from  $(u_l, v_l)$ , and to which  
 685 we can apply the same re-writing as in the case of  $tw(G) \leq 3$ .

686  $\blacktriangleleft$