Algorithmique paramétrée exacte pour la bioinformatique structurale des ARNs

Bertrand Marchand

INSTITUT POLYTECHNIQUE DE PARIS

NNT : 2023IPPAX072

Thèse de doctorat

Exact Parameterized Algorithmics for Structural RNA Bioinformatics

Thèse de doctorat de l'Institut Polytechnique de Paris
préparée à l'École Polytechnique

École doctorale n°626 École doctorale de l'Institut Polytechnique de Paris (EDIPP)
Spécialité de doctorat : Informatique

Thèse présentée et soutenue à Palaiseau, le 27/09/2023, par

BERTRAND MARCHAND

Composition du Jury :

Christian Komusiewicz Professeur, Friedrich-Schiller-Universität Jena	Président
Marie-France Sagot Directrice de recherche, INRIA	Rapporteure
Peter Stadler Professeur, Universität Leipzig	Rapporteur
Michal Ziv-Ukelson Professeure, Ben Gurion University	Rapporteure
Julien Baste Maître de conférence, Université de Lille	Examinateur
Florian Sikora Maître de conférence, Université Paris-Dauphine	Examinateur
Laurent Bulteau Chargé de recherche, CNRS	Co-superviseur
Yann Ponty Directeur de recherche, CNRS	Directeur de thèse

Abstract

RNAs are one of the fundamental building blocks of life, along with DNA and proteins. If they are mostly known as a mere intermediate in the synthesis of proteins (*messenger* RNAs), they may also act directly as RNA to perform a wide variety of functions (catalysis, expression regulation…). For these *non-coding* RNAs, the *folded structures* they adopt are crucial.

Both RNA sequences and structures display an inherently *combinatorial* nature: sequences are words over `A,U,G,C`, while structures mainly consist of `A-U`, `G-C` and `G-U` base-pairs. Several fundamental computational problems involving functional RNAs are therefore naturally expressed in the language of discrete mathematics. Such problems include RNA FOLDING (what is the preferred structure of a sequence?), RNA DESIGN (how do I find a sequence that would fold into a given structure?) or RNA ENERGY BARRIER (is there a feasible transition between two structures?). Some of these fundamental problems are NP-hard, but still need to be solved by RNA bioinformaticians in practice, either to better understand biological systems or for the development of RNA therapeutics (e.g. COVID19 vaccines). These potential applications, combined with the ever-increasing amount of sequencing data available, mean there is a dire need for efficient methods.

The philosophy of this PhD thesis is to explore the possibility of applying *parameterized algorithmics*, a relatively recent and very dynamic field of algorithmic research, to hard structural RNA bioinformatics problems. A particular focus is given to *graph formulations* and graph *width measures* as parameters.

Résumé

Les ARNs (Acides Ribo-Nucléiques) constituent, avec l'ADN et les protéines, l'un des blocs élémentaires sur lesquels sont construits tous les systèmes biologiques. Si ils sont surtout connus comme étant de simples intermédiaires dans la synthèse de protéines (ARNs messagers), ils peuvent aussi agir directement en tant qu'ARN, et remplir alors des rôles très variés (catalyse, régulation de l'expression de gènes…). Pour ces ARNs dits *non-codants*, la *structure de repliement* qu'ils adoptent est cruciale.

À la fois les séquences et les structures d'ARN présentent un aspect intrinsèquement *combinatoire*: les séquences sont des mots sur l'alphabet `A,U,G,C`, tandis que les structures sont principalement constituées de paires de bases `A-U`, `G-C` et `G-U`. Plusieurs problèmes fondamentaux impliquant les ARNs non-codant sont par conséquent naturellement exprimés dans le language des *mathématiques discrètes*. Ces problèmes incluent le *repliement* (Quelle est la structure préférentielle d'une séquence donnée ?), le *design d'ARN* (Comment trouver une séquence se repliant selon une structure spécifiée en entrée ?) ou le calcul de *barrières d'énergie* (Y'a-t-il une transition entre deux structures susceptible de survenir spontanément ?). Certains de ces problèmes fondamentaux sont NP-difficile, mais les bioinformaticiens de l'ARN doivent tout de même les résoudre quotidiennement, soit pour mieux comprendre les systèmes biologiques naturels, soit pour le développement de *thérapies à base d'ARN* (dont les vaccins contre le COVID19 sont un exemple). Étant donné également les quantités toujours plus grandes de données de séquençage à traiter, il y a un besoin croissant de méthodes algorithmiques efficaces pour les problèmes mentionnés ci-dessus.

La philosophie de cette thèse de doctorat est d'explorer les possibilités d'application de l'algorithmie *paramétrée*, un domaine relativement récent et très dynamique de la recherche algorithmique, à des problèmes difficiles de bioinformatique des ARNs. Une attention particulière est donnée aux formulations en termes de *graphes*, et à des paramètres de *largeurs de graphes*.

Acknowledgements/Remerciements

I would like to start by thanking all members of the Jury for accepting to evaluate this PhD thesis, for their helpful comments on both the manuscript and the defense, and for their thought-provoking questions. Special thanks go of course to the reviewers Marie-France Sagot, Peter Stadler and Michal Ziv-Ukelson (particularly given the reviewing window matching almost *exactly* with summer holiday times); and to Christian Komusievicz for his service as president of the Jury, which involved a trip using both Deustsche Bahn *and* SNCF to come physically for the occasion.

Then, my most special thanks go to Laurent and Yann, for their guidance, patience and dedication throughout these three years. This PhD has been an exceptionnally fulfilling and transformative experience for me. I owe it to them, and to the quality of the interactions we had over these three years. They are examples to follow, both from a research perspective, and in how they *care* about the people they work with.

My path towards research in Computer Science has not been linear. Therefore, I want also to thank them for their openmindedness when considering a PhD application from an R&D engineer with an unusual profile, and giving it a go. The same goes for the PhD selection committee of EDIPP for the year 2020, whose composition I do not precisely know, which allocated me a scholarship.

To continue, I would like to thank all the people I have worked with during these three years, especially the members of the Amibio team I overlapped with: Hua-Ting, Sarah, Sebastian, Taher, and Théo. Whether by coming to the office or by turning on Discord/Zoom, it has always been a pleasure ! Many thanks as well to Aïda and Manuel for welcoming me in Sherbrooke for a research internship, that I liked so much I decided to stay for a post-doc. Speaking of Québec, warm thanks to Vlad, for being so welcoming, for so many discussions, including advice on both post-docs and (a geographically *wide* set of) restaurants.

To say that I do not regret leaving the private sector for Computer Science research is an understatement. A primary argument for this is the *genuine* aspect of both research discussions and informal meetings with the nice people that make up this community. In this spirit, I would like to thank many people I dicussed with at some point during the PhD, such as the members Benasque community, the partners of the DECRyPteD ANR, my fellow supervisors of exercise sessions for CSE201 and INF411 at École Polytechnique, and the participants and organizers of the conferences and summer schools I took part in.

I also want to express my gratitude to the people who made this manuscript better by proof-reading it and making comments. This is adressed to Laurent and Yann of course, but also my brother Lilian, my friends François, Guillaume, and Julien, and my life-partner Gabrielle.

Merci également à toutes celleux qui on fait le trajet jusque Palaiseau pour venir assister à la soutenance de cette thèse: Alexis F, Alexis M, Emmanuel, Alain, François, Guillaume, Julien, Laurie, Lucie, Mélanie, Raphaël, Thomas, ainsi que mes parents, mes frères, et Gabrielle. Merci aussi à celles et ceux qui, sans parvenir à venir à la soutenance, ont ensuite pris part à la célébration: Annabelle, Brigitte

pour son montage de la chanson, Camille, Charlotte, Dimiri, Élodie, Johann, Marie-Thècle et Yaëlle, Nam et Théophile.

Pour finir, et sur une note plus personnelle, j'aimerais remercier ma famille et mes ami.e.s pour leur soutien dans cette aventure. Ce soutien a pris des formes multiples, allant de la simple disponibilité pour discuter de tout et de rien autour d'une bière ou au pied d'un mur d'escalade, à l'expression d'un enthousiasme vis-à-vis de cette idée saugrenue de retourner en recherche. Plus particulièrement, merci à mes parents et à mes frères d'être ce refuge de bienveillance et de gentillesse. Merci à ma maman pour son exemple de professionalisme, et d'abnégation pour faire grandir un projet auquel on croît.

Et enfin, parce que quand nous sommes tous les deux tout semble tellement plus facile, merci, pour tout, à ma partenaire de vie Gabrielle.

Eh approche
Écoute, hoche
la tête si t'accroche
pour ta famille et tes proches
va grapher dans la roche !

Revisite de *Gravé dans la roche* (Sniper) par
un collectif composé notamment de
Brigitte, Élodie, Gabrielle, Guillaume,
Johann et Julien.

Contents

Chapter 1

Introduction

The purpose of this chapter is to give the reader prerequisite elements of structural RNA bioinformatics and parameterized complexity. A particular focus is given to *structured* RNAs, and the computational problems that emerge when studying them. The originality of this thesis is to attempt to tackle them with parameterized approaches based on *width measures*, which are also introduced here.

Organization. This introduction is roughly split in two, with the first half covering RNA bioinformatics, and the second parameterized algorithmics. Some of the most technical content has been isolated in gray-shaded boxes, that may be skipped in a first reading. Almost all paragraphs have names, with those of **this color** indicating a higher level heading ($\sim$ sub-sub-section), potentially containing several paragraphs whose titles are typeset with **that color**.

1.1 RNA bioinformatics

1.1.1 RNA basics

Ribo-Nucleic Acid (RNA). RNA is a category of biopolymers, and one of the most fundamental building blocks of life. It constitutes one of the great families of modular biological macro-molecules, along with DNA or proteins. In virtually any life form, it is one of the carriers of genetic information, and even the *only* carrier in the case of RNA viruses (HIV, SARS-COV2,...). However, as we shall see in more depth in this section, it also assumes a variety of other roles (catalysis, gene regulation,...) thanks to its structural properties. This versatility, along with the existence of the RNA-based life forms mentioned above, have even led to the hypothesis of an "RNA world" as a potential origin of life [1]. From a more applicative point of view, in medical research, the functional flexibility of RNA is being exploited with the development of *RNA therapeutics* [2, 3], of which RNA vaccines [4] are an example.

Basic structure. From a biochemical point of view, an RNA molecule consists of an oriented chain of molecular units called *nucleotides*. 4 different units are allowed, denoted by `A,U,G,C` (Adenine,

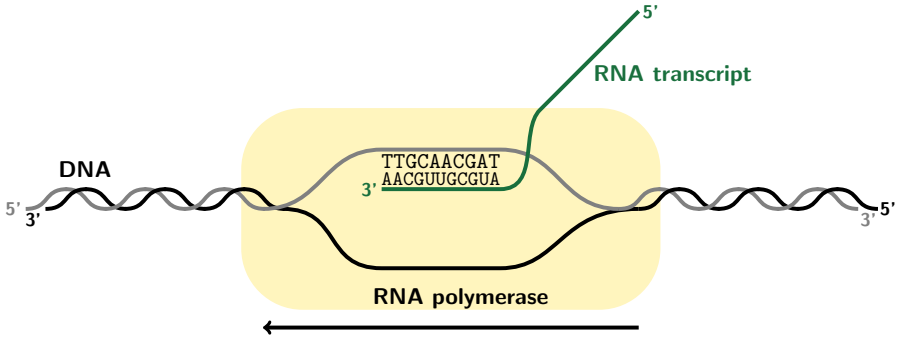


Figure 1.1: RNA molecules (often also called RNA *transcripts*) are synthesized from DNA by a molecular complex called RNA polymerase. As it travels along the DNA on one of its strands, it appends to the RNA transcript the complement of the nucleotide it reads on the strand. All nucleotide strands are oriented, as denoted by the 5' and 3' at their extremities. These denominations originate from the fine biochemical structure of nucleic acids. Synthesis and reading of nucleic acids occur in a specific direction only (3' to 5' and 5' to 3', respectively)

Uracil, Guanine and Cytosine). The variable part of a nucleotide is called its *base*. Nucleotides also have a constant part, identical in `A`,`U`,`G` and `C`, composed of a *sugar* and a *phosphate group*. Crucially, it is through this constant part that nucleotides are connected to form an RNA molecule, leaving the bases free to interact (potentially with bases from other nucleotides).

RNAs are synthesized as copies (or *transcripts*) of portions of DNA, by a molecular machinery called *RNA polymerase* that travels along one DNA strand. This process, called *transcription*, is illustrated on Figure 1.1. DNA is indeed famously composed of two *complementary* nucleotide strands forming a double helix. Each strand is a sequence of four possible nucleotides (`A`,`T`,`G`,`C`), and each nucleotide is *paired up* with a complementary nucleotide on the other strand. An Adenine (`A`) is always paired with a Thymine (`T`), and a Guanine (`G`) with a Cytosine (`C`). `A`-`T` and `G`-`C` constitute the so-called *Watson-Crick* base-pairs, consisting of hydrogen bonds connecting the two molecules. This structure is illustrated on Figure 1.2 (left).

As one may have noticed, Uracil (`U`) replaces `T` in RNA strands, compared to DNA. Importantly, `U` can still pair up with `A` to form a Watson-Crick base-pair. Actually, it can do more, and also pair up with `G` to form a `G`-`U` base pair, traditionally called *wobble pair*[1]. An RNA transcript being a single strand, it can therefore fold onto itself to allow for such base-pairs to form between its nucleotides. These nucleotides being brought together in space by chemical bonds, the result is a potentially complex 3D structure. A simple example, along with an illustration of all possible base-pairs, is given on Figure 1.2 (right). **These *folded structures* or *conformations*, often abstracted as *graphs*, play a central role in all the computational problems studied in this PhD thesis.**

[1] Actually, all pair-wise interactions (`A`-`A`,`G`-`A`,...) are virtually possible [5]. But it is `G`-`U`,`A`-`U`,`G`-`C` that is usually retained as the *canonical* RNA base-pair set [6].

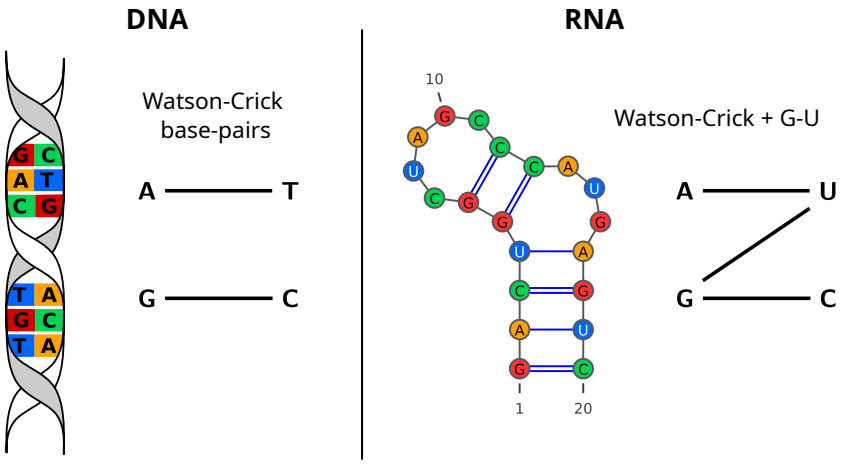


Figure 1.2: (left) DNA molecules are composed of two complementary strands of nucleotides. Each nucleotide of a strand is paired up with a complementary nucleotide on the other strand, following the Watson-Crick base pairs (A-T, G-C). (right) An RNA molecule is a single strand of nucleotides. Watson-Crick base-pairs can still form between nucleotides of the strand, as well as G-U base-pairs. The formation of these base-pairs is what may give complex spatial conformations to RNA molecules. This structure is of critical importance to *functional* RNAs (i.e., RNAs that act directly as such in biological systems).

Functional RNAs. RNA is mostly known for its role in the "central dogma of molecular biology" (Figure 1.3 (b), black arrows) as *messenger* RNA, a simple intermediate in the synthesis of proteins. In a nutshell, the portion of DNA coding for a protein is first *transcribed* as a messenger RNA, before being *translated* into proteins by the ribosomes.

But some RNA transcripts are never translated into proteins, and take up a biological function directly as RNAs. Generically dubbed *non-coding RNA* or *functional RNA*[2], they constitute in fact most (if not the overwhelming majority) of RNA transcripts in complex organisms [8]. Their roles are extremely varied (Figure 1.3, green arrows), ranging from chemical reaction catalysis to gene regulation and aspects of protein synthesis. Famous example include *transfer RNAs* (responsible for fetching amino-acids corresponding to a given codon in protein synthesis), *riboswitches* [9] (which can control the expression of a gene depending on the presence of a specific chemical) or *ribozymes* [10] (catalysis). The ribosomes themselves, where proteins are synthesized, are composed of *ribosomal RNA*, with RNA chains of thousands of nucleotides.

ncRNA families. A common feature to non-coding RNAs is that their functions critically depend on the adoption by the molecule of one or several structural conformations. This results in evolutionary pressure towards structure conservation, and therefore a restriction of possible genetic mutations to

[2]To be precise, the question of what fraction of non-coding RNA (i.e., RNA not translated into proteins) is functional RNA (i.e., has an actual biological role) is still a matter of debate [7].

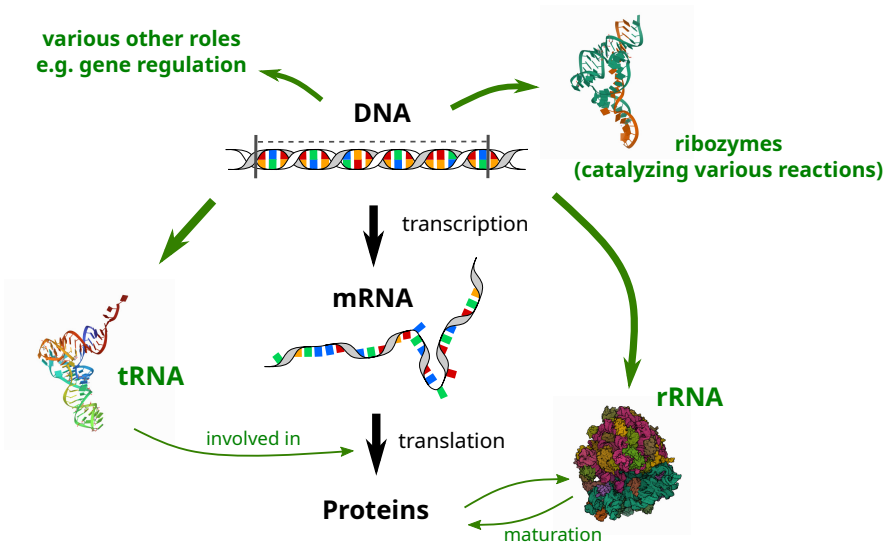


Figure 1.3: RNAs are intermediates in the synthesis of proteins (black arrows), but not only: they act directly as RNAs for a variety of functions, in which the conservation of their folded 3D structure, more than their sequence, is critical. Examples include *transfer RNA* (tRNA) and *ribosomal RNA* (rRNA), both involved in protein synthesis, as well as *ribozymes* (catalysis) or *riboswitches* (gene regulation).

an exploration of a set of compatible sequences. A good example of structure-preserving mutation is a substitution of one or both ends of a base-pair with another set of compatible nucleotides (e.g.`A-U` to `G-C` or `G-U`). For a base-pair (i, j), such a mutation of site j to keep a base-pair with a mutated site i is typically called *compensatory mutation*.

Considering a basic model for mutations, consisting in *point-wise mutations* (a nucleotide replaced by another), *insertions* and *deletions*, a practical way of representing a set of nucleotide sequences originating from a common ancestor is a *multiple-sequence alignment* (MSA). In a multiple sequence alignment, all input sequences are possibly augmented with *gap symbols* (usually `-') to represent insertions/deletions of nucleotides. When aligning RNA sequences that are variants of the same structured RNA, the alignment is usually annotated with a *consensus structure*, which makes compensatory mutations apparent. An example of MSA is given on Figure 1.4, along with secondary structure annotation and examples of compensatory mutations. Different sequences corresponding to the same ncRNA are called *homologs*. The RFAM database [11] (`https://rfam.org/`) is a collection of such families of homologs, along with curated MSAs. As of version 14.9 (November 2022), it counts 4108 families. Its data constituted one of the main testing ground for the algorithms developed in this PhD thesis.

Computational questions. When studying the structural properties of RNA, several questions naturally come up.

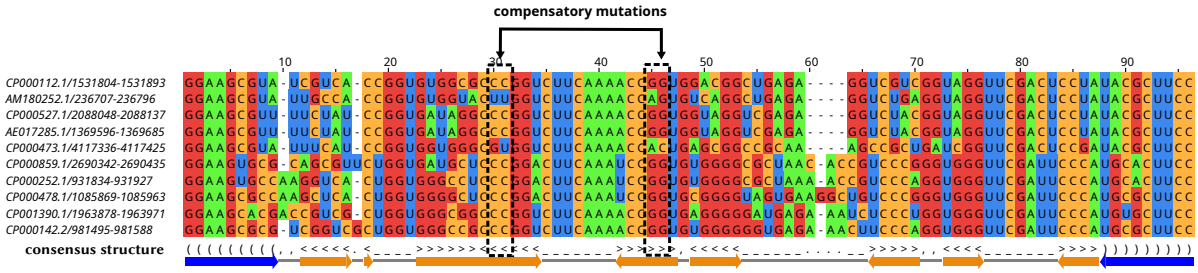


Figure 1.4: Example of a multiple sequence alignment with consensus structure annotation. This particular example is a subset of the seed alignment of the rfam family RF01852 (transfer RNA). Columns with compensatory mutations preserving the consensus structure are highlighted. This alignment picture was generated using `Jalview` [12].

- **Folding:** The most immediate is perhaps the prediction of the (set of) preferred structure(s) of a given RNA, which is known as the *folding* problem [13]. To solve it algorithmically, one must enumerate a *class* of structures, and output the *most stable* one for the input sequence, i.e. the one with the *lowest free energy*.

- **Design:** Conversely, for instance in medical applications, the question may come up of *designing* a sequence that folds preferably in a prescribed structure [14]. The algorithmic task is to find a sequence (over $\{\texttt{A},\texttt{U},\texttt{G},\texttt{C}\}$) of *low energy* for the input structure, and ideally also of *higher energy* for alternative structures.

- **Kinetics:** From a more dynamical point of view, one may also wonder whether an RNA molecule reliably always folds the same way, or is likely to transition between several structures of comparable stability. This raises questions of *kinetics* and *energy barriers* [15]. Algorithmically, it involves exploring sets of *reconfiguration pathways* to identify feasible ones.

- **Alignment:** As touched upon above, a fundamental task in the study of ncRNAs is the clustering of sequences into *families of homologs*. To decide whether an input sequence belongs to a given ncRNA family, the computational question is whether it *aligns* well with the family, in a way that preserves the *consensus structure* (Figure 1.4). A first step is typically to compute alignments of *pairs* of sequences, which consists in finding a *mapping* between the positions of the two sequences.

RNA bioinformatics emerged as a field to try and provide efficient methods to tackle such questions. To state them more precisely, and review existing approaches from the literature (along with their limitations), one needs to define *energy models* and some *classes* of structures. This is the purpose of the next section. The computational complexity of the questions formulated above will typically depend on the choice of energy model and class of structures to restrict the search to.

1.1.2 RNA structures

The biochemical properties of RNAs induce constraints and preferences as to the sets of base-pairs adopted by a sequence. This section defines structural units typically used when discussing RNA structures, as well as a hierarchy of classes of structures of increasing complexity.

Notations: sequence and list of base-pairs. An RNA sequence S is an element of $\{\texttt{A},\texttt{U},\texttt{G},\texttt{C}\}^*$. Throughout this document, *structured RNA* will typically be denoted by $(S, \mathcal{A})$, with S a sequence of length N and $\mathcal{A}$ a set of pairs (also called arcs) (i, j) such that $0 \leq i, j \leq N-1$. $(S, \mathcal{A})$ will also be called *arc-annotated sequence*. Given a sequence S, the sub-sequence corresponding to an interval of positions going from i to j (both included) is noted $S[i \dots j]$.

Hierarchical folding. The folding process of RNA is recognised to have a *hierarchical* aspect [6]. As a *primary structure*, the backbone of the sequence itself is never broken during the lifespan of an RNA molecule. *Secondary* and *tertiary* structural elements then come on top. The overall picture is that secondary elements form at a faster timescale, and are more stable, than tertiary elements.

This hierarchical aspect gives a natural way of defining classes of structures of increasing complexity. However, some discussion may arise as to what patterns count as secondary or tertiary. A prominent example is the case of *pseudoknots*, as discussed further down.

Secondary structures. We adopt in this thesis the following definition for secondary structures.

Definition 1 (Secondary structures)**.** A set $\mathcal{A}$ of base-pairs is a *secondary structure* if each position is involved in at most one base-pair.

Note that in this definition, a secondary structure may exhibit *crossings*, i.e. there can be two base-pairs (i, j) and (k, l) such that $i < k < j < l$ or $k < i < l < j$. Such base-pairs are also said to be in *conflict* or to form a *pseudoknot*. An illustration of a crossing is given on Figure 1.6 (a).

It is quite typical in the literature to designate as *RNA secondary structures* what is here called *conflict-free secondary structures*. The difference is whether pseudoknots are considered as *tertiary structure* elements or not. Considering this wider definition of secondary structure is not unheard of ([16]) and is justified by the indications that pseudoknots occur with high prevalence in natural RNAs [17]. It is mainly for computational reasons, as we shall see in the next section, that crossings are typically excluded from structure sets.

Notations: non-crossing base-pairs relations. If two base-pairs (i, j) and (k, l) do not cross, then either one is nested in the other (e.g. $i < k < l < j$) or they are one after the other (e.g. $i < j < k < l$). We adopt the following notations for these two settings: we write $(k, l) \subset (i, j)$ if $i < k < l < j$ (nesting) and $(i, j) \parallel (k, l)$ if $i < j < k < l$ (parallel).

Notation 1. *nested* ($\subset$) *and parallel base-pairs* ($\parallel$)

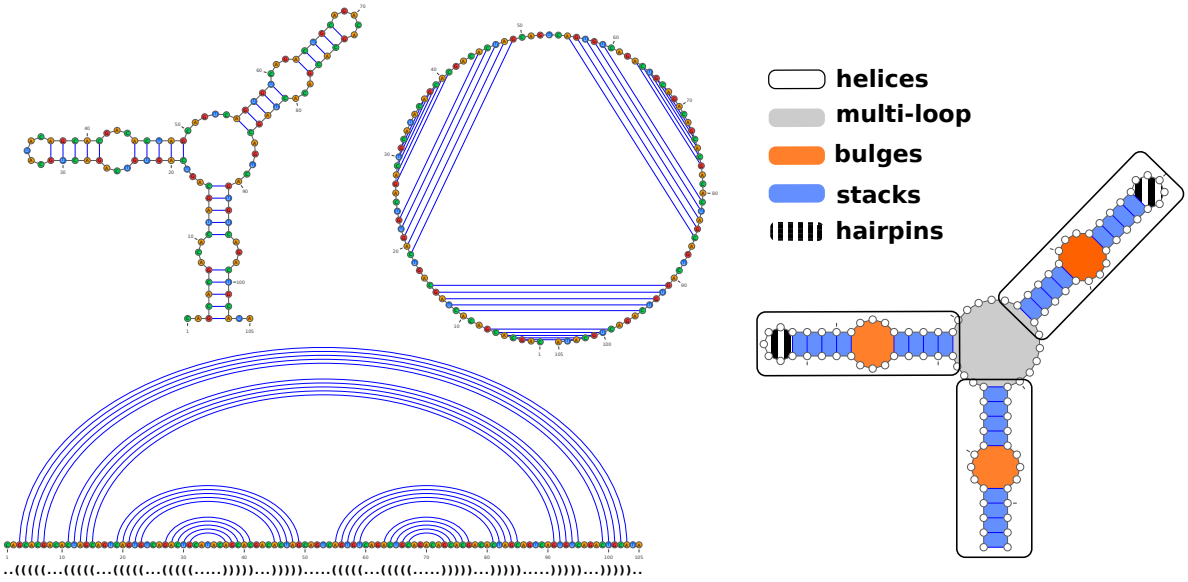


Figure 1.5: A conflict-free secondary structure is a set of base-pairs involving at most one nucleotide at a time and no crossings. It can be represented as an outer planar graph (top left), non-crossings chords of a circle (center) or a well-parenthesized string (bottom). As the main class of RNA structures studied and represented in the literature, its structural elements have standard names (helices, bulges, stacks, hairpins), sketched on the right.

Conflict-free secondary structures. The most studied class of RNA structures in the literature is undoubtedly *conflict-free secondary structures*. Thanks to the absence of conflicts, each base-pair naturally defines an inside and an outside, which makes this class of structures very practical to work with from an algorithmically. A lot of foundational algorithmic results for RNA Bioinformatics are set within this class [18, 19, 20], as we shall review in the next section.

Thanks to the absence of crossings, conflict-free secondary structures may be represented in a *dot-bracket notation*, i.e a string composed of well-nested parentheses `()` representing base-pairs, and dots '`.`' for unpaired positions. An example of a dot-bracket notation is given on Figure 1.5, along with other possible representations for conflict-free secondary structures, as outer planar graphs and non-crossing sets of chords of a circle. The standard nomenclature for some structural elements (bulge, hairpin, helix...) are also given.

Pseudoknotted secondary structures. When a secondary structure exhibits pseudoknots, as in the example on Figure 1.6 (b), several parenthesis systems (`()`, `[]`, `{}`...) are needed to represent base-pairs. As we shall see, taking pseudoknots into account drastically increases the computational complexity of many problems. Indeed, it removes the possibility of formulating algorithms as Dynamic Programming scheme over intervals of the input sequence. Famous pseudoknotted patterns found in

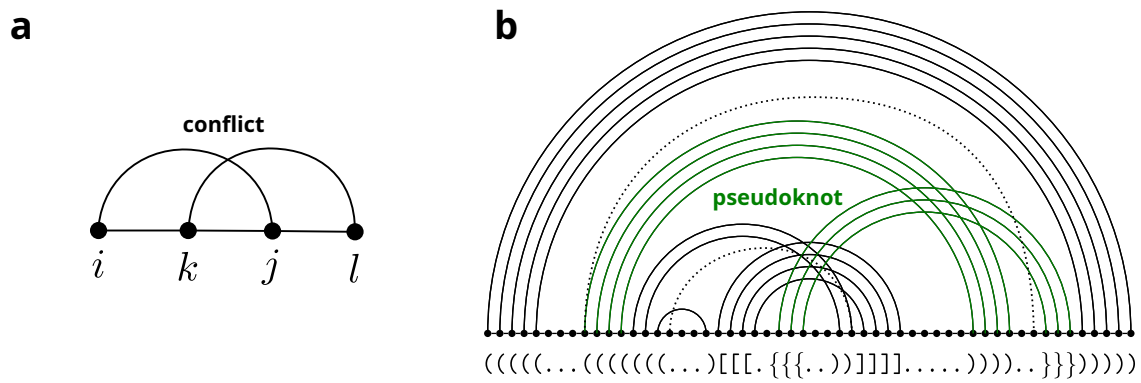
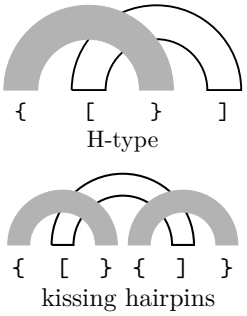


Figure 1.6: (a) Illustration of the definition of crossing base-pairs. (b) A pseudoknotted secondary structure (solid edges) and additional tertiary interactions (dashed edges). A possible dot-bracket notation is also displayed. The potential additional tertiary interactions could correspond to non-canonical pairing, that could involve already paired nucleotides, putting the graph outside the class of secondary structures.

natural RNAs include "H-type" pseudoknots (`{[}]` pattern) or "kissing hairpins" (`{[}{]}` pattern). See margin for illustrations.



Tertiary structure. Nucleotides in RNA molecules have also been shown to form so-called "non-canonical" interactions [5], with evidence that they play a critical role in the adoption of the full conformation [21]. As displayed on Figure 1.6 (b) with dotted edges, non-canonical interactions may involve nucleotides that are already part of a base-pair. Examples include base-pairs such as `G-A`, or interactions involving more than two nucleotides (multiplets) [22, 5].

Formally, within this thesis, a "tertiary structure" is any arbitrary set of base-pairs, because the only defining constraint for secondary structures of having only one base-pair per nucleotide is lifted. However, a reasonable practical assumption is that any nucleotide is not involved in more than a few base-pairs, which is the case in standard nomenclatures for non-canonical base-pairs [5, 23].

Lack of computational tools. As we shall cover in Section 1.1.4, taking pseudoknots into account generally increases the complexity of RNA bioinformatics problems. This is true in particular for *folding* and *structure-aware alignment*. A consequence is that pseudoknots remain challenging to take into account in practical Bioinformatics projects. For instance, pseudoknotted families in the `Rfam` [11] database are still partially curated *manually*. When it comes to folding, the most popular software programs [24, 25, 26] do not support pseudoknots. One could also mention studies of *RNA-RNA interactions* [27], which require computational tools capable of taking crossing into accounts. **This PhD thesis is in part motivated by the need to develop algorithmic methods capable of taking pseudoknots into account.**

1.1.3 Energy models

RNA structure and Boltzmann distribution. RNAs adopt their folded structures through the formation of *base pairs*, chemical bonds that bring nucleotides together in space. These base pairs form spontaneously, as they lower the *free energy* of the molecule, making it more stable. The probability for an RNA to adopt a given structure $\mathcal{S}$ depends on this free energy $E(\mathcal{S})$ following, in the *thermodynamic equilibrium*, a *Boltzmann distribution* $\mathbb{P}(\mathcal{S}) \propto e^{-\beta E(\mathcal{S})}$. The structure having *minimum free energy* is then the most probable. A multitude of *energy models*, associating a set of base pairs to an energy, are used in RNA bioinformatics, with various levels of complexity.

This section reviews classic energy models for RNA structures. Combined with the structure classes defined in Section 1.1.2, it will allow us to state the computational complexity of classic RNA structural bioinformatics problems in Section 1.1.4.

Base-pair - or "Nussinov" - model. As a very basic rule of thumb, the more base-pairs a structure has, the more stable it is. This is captured by a simple energy model associating a weight of -1 to every base-pair, as defined below.

Definition 2 (Nussinov model)**.** Given $\mathcal{A}$ a set of base-pairs over an RNA sequence S, the *Nussinov energy* associated to it is:

$$E_{\texttt{\#bps}}(\mathcal{A}) = \sum_{(i,j)\in\mathcal{A}} -1 = -|\mathcal{A}|$$

The name of this model stems from Nussinov's seminal algorithm [18], capable of producing, given a sequence S as input and in time $O(|S|^3)$, a *conflict-free secondary structure* $\mathcal{S}$ minimizing $E_{\texttt{\#bps}}$. It will be explained in more detail in Box 1, page 21.

Weighted base-pairs. A slight generalization of this model, accounting for the fact that some base-pairs have a higher energetic contribution compared to others (`G-C` is stronger than `A-U` and `G-U`) is a *weighted base-pairs* model.

Definition 3 (weighted base-pairs model)**.** Given $\mathcal{A}$ a set of base-pairs over an RNA sequence S, and negative weights $w_{\texttt{GC}}$, $w_{\texttt{AU}}$ and $w_{\texttt{GU}}$, the weighted base-pairs energy associated to it is:

$$E_{\texttt{w-bps}}(\mathcal{A}) = \sum_{(i,j)\in\mathcal{A}} w_{S[i]S[j]}$$

Nussinov's algorithm can easily be modified to support this model with the same complexity.

Turner energy model. The two models defined above can be applied to any of the structure classes of the previous section. However, it is widely accepted that they miss out on critical energy contributions: *stacking* energies between two nested base-pairs (i, j) and $(i + 1, j - 1)$. Stacking terms encourage the formation of perfect stacks of base-pairs called *stems*, which gives rise to much more realistic structures.

Of lesser importance, but also ignored by base-pair models, are energetic contributions of the composition of *loops* and *bulges* (see Figure 1.5 (right)). Taking both into account yields the Turner energy model [28], the gold standard of RNA energy models. Let us start with the *stacking energy model*, easily defined in terms of base-pairs only:

Definition 4 (stacking energy model)**.** Given $\mathcal{A}$ a set of base-pairs over a sequence S, and weights $w_{xy,zt}$ for every possible nucleotide combination in two stacked base-pairs (x, y) and (z, t):

$$E_{\texttt{stacking}}(\mathcal{A}) = \sum_{(i,j)\in\mathcal{A} \text{ s.t. } (i+1,j-1)\in\mathcal{A}} w_{S[i]S[j],S[i+1]S[j-1]}$$

In order to formulate the Turner energy model, let us give a definition for loops in sets of base pairs. Indeed, the Turner energy model consists in the stacking model augmented with *loop* energy terms. Using Notation 1, it reads:

Definition 5 (loop in an RNA structure $\mathcal{A}$)**.** A loop of $\mathcal{A}$ is a set of base-pairs $\{(i_k, j_k)\}_{0\leq k\leq p}$ such that:

- $\forall k \in [1, \dots, p]$, $(i_k, j_k) \subset (i_0, j_0)$
- $\forall k, l \in [1, \dots, p]$, $(i_k, j_k) \parallel (i_l, j_l)$
- for any other base-pair $(x, y) \in \mathcal{A}$ such that $(x, y) \subset (i_0, j_0)$, $\exists k$ such that $(x, y) \subset (i_k, j_k)$

In other words, a loop consists of an overarching base-pair (i_0, j_0) containing a (potentially empty) set of parallel base-pairs $\{(i_k, j_k)\}_{1\leq k\leq p}$. The *nucleotide content* $c(\ell)$ of a loop is then

$$c(\ell) = S[i_0 \dots i_1] \cup \left(\bigcup_{1\leq k\leq p-1} S[j_k \dots i_{k+1}] \right) \cup S[j_p \cdots j_0]$$

p is called the *order* of a loop. The definition of a loop is illustrated on Figure 1.7. Note that *stacked base-pairs*, as in Definition 4, are particular examples of loops (with $p = 1$).

Given this definitions, the Turner energy model is formulated as such:

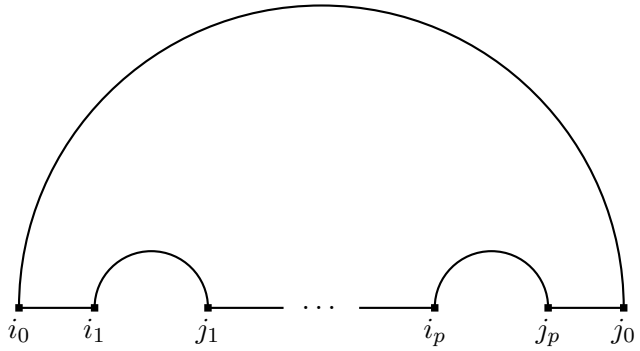


Figure 1.7: Illustration of the definition of a loop.

Definition 6 (Turner energy model)**.** For $\mathcal{S}$ a set of base-pairs, and a weight function w associating contributions to nucleotide contents of loops, the *Turner energy* is:

$$E_{\texttt{Turner}}(\mathcal{A}) = \sum_{\ell \text{ loop}} w(c(\ell))$$

Values of $w(c)$ for a nucleotide content c have reference *tabulated* values up to a certain loop size. For large loop sizes, and $p > 2$ (multiloops), some simplifying extrapolation approximations are standard [19]. The rationale behind them is both the difficulty of tabulating the corresponding energy contributions experimentally, and the acceleration they allow in folding algorithms (see section 1.1.4).

In the case of pseudoknot-free secondary structures (Figure 1.5), loops, bulges and stacks constitute a full coverage of the set of *faces* delineated by the structure taken as an outerplanar graph (as one can see from the coloring on Figure 1.5 (right)).

Turner extensions for pseudoknots. In the case of pseudoknotted structures, the Turner energy is still defined, but the coverage property described above does not verify. Some extensions of the Turner energy model with pseudoknot-specific terms (such as pseudoknot creation penalties) have been developed, particularly in the context of *folding algorithms* [29, 30, 31] (see Section 1.1.4.1 below). However, the Turner model still constitutes the core of these models, with *stacking* terms (Definition 4) being typically considered as the main qualitative difference with respect to weighted base-pairs models.

1.1.4 Computational problems

As argued above, the study of functional RNAs immediately suggests several fundamental computational problems. This section defines formally some of them, which have constituted the targets for

structure class $\mathcal{C}$ →

energy model $\mathcal{E}$ ↓

RNA FOLDING	conflict-free	PK sec	tertiary
Nussinov (Def. 3)	**P** - [18] & Proposition 1 [32]		
Stacking (Def. 4)	**P** [19]	**NP-hard** [33, 34]	
Turner (Def. 6)			

Table 1.1: The hardness of RNA FOLDING depends on the class of structures we restrict the search to, and on which energy model is optimized for.

algorithmic developments in this PhD thesis. For each of them, a review of results from the literature, such as hardness proofs or classic algorithms, is given. Computational hardness, as we shall see, crucially depends on the choice of structure class (among the possibilities described in section 1.1.2) and energy model (section 1.1.3).

1.1.4.1 RNA folding

A central hard problem in RNA bioinformatics is RNA FOLDING, the task of finding the minimum-free energy structure for an input sequence. We formally state it below.

Problem 1 (RNA FOLDING)**.**
Input: Sequence $S \in \{\texttt{A},\texttt{U},\texttt{G},\texttt{C}\}^*$
Output: A set $\mathcal{A}$ of base-pairs within a class of structure $\mathcal{C}$, minimizing an energy function $\mathcal{E}$

The computational complexity of RNA FOLDING is summarized on Table 1.1, depending on the choices of $\mathcal{C}$ and $\mathcal{E}$. A lot of the references cited in this Table are foundational results in RNA bioinformatics.

Nussinov's algorithm. One of these seminal results is Nussinov's dynamic programming algorithm for RNA FOLDING in the weighted base-pair energy model and conflict-free secondary structures [18]. It is a prime example of a dynamic programming (DP) strategy applied to an RNA bioinformatics problem. Given the ubiquity of such approaches in the field, we provide a short presentation of Nussinov's algorithm in Box 1.

Weighted base-pairs model and pseudoknots. Perhaps surprisingly, RNA FOLDING remains polynomial in the weighted base-pair energy model for all structure classes (first line of Table 1.1). The pseudoknotted case is handled with a reduction to maximum weighted matching on the "compatibility graph" of a sequence, as formulated below.

Definition 7 (compatibility graph)**.** Given a sequence S, the *compatibility graph* $G_{\texttt{C}}(S)$ is a graph where each nucleotide is a vertex, and two nucleotides (i, j) are connected if

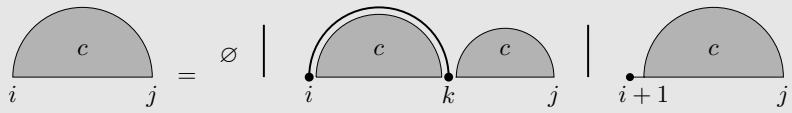
$$(S[i], S[j]) \in \{(\texttt{A},\texttt{U}), (\texttt{U},\texttt{A}), (\texttt{G},\texttt{C}), (\texttt{C},\texttt{G}), (\texttt{G},\texttt{U}), (\texttt{U},\texttt{G})\}$$

Box 1. (Nussinov's algorithm [18])
As any dynamic programming approach (see [35]), it relies on the definition of *sub-problems* connected together by *recursive equations*. Intuitively, problems are cut into sub-problems that are solved independently and then *merged* into a global solution. In the context of RNA bioinformatics, sub-problems often correspond to *intervals* $[i, j]$ of an input sequence S. Solutions to these sub-problems are stored (cached) in a *DP table* c so that they are computed only once. In the case of Nusinov's algorithm, $c[i, j]$ contains the minimum possible energy value $E_{\texttt{w-bps}}(\mathcal{A})$ over all possible conflict-free secondary structures $\mathcal{A}$ for the sub-sequence $S[i \dots j]$. Defined this way, it verifies the following:

$$c[i,j] = \min \begin{cases} 0 & \text{if } j - i \leq \theta \text{ (terminal case)} \\ \min_{k \in [i+\theta, j]} \left(w_{S[i]S[k]} + c[i+1, k-1] + c[k+1, j] \right) & \text{(base-pair creation)} \\ c[i+1, j] & (i \text{ unpaired}) \end{cases}$$

With θ an enforced minimum possible distance between the two ends of a base-pair (usually $\theta \simeq 3 - 5$). These equations may then be solved with either a bottom-up iteration over intervals of growing sizes (i.e., over sub-problems with an ordering that guarantees $c[i+1, j]$, $c[i+1, k-1]$ and $c[k+1, j]$ are computed before $c[i, j]$) or a top-down memoization approach (i.e., a recursive implementation with caching). The overall complexity is $O(n^2)$ in space, and $O(n^3)$ in time.
To be specific, the recursive relations above are a weighted generalization of the original formulation of Nussinov's algorithm, which intended to minimize $E_{\texttt{\#bps}}$. The only difference is the use of weights dependent on the base-pair which is created. Other early works containing similar recursive relations for weighted energy models include [36].

Associated grammar. A graphical representation of the dynamic programming scheme above could be:



This decomposition scheme relies on the fact that, when a conflict-free secondary structure is not empty, then either the first position in the sequence is paired to some k, or it is unpaired. When it is paired to some $k \in [i + \theta, j]$, no other base-pair can cross (i, k). This decomposition scheme can directly be seen as *production rules* for a grammar producing legal structures. Writing structures in dot-bracket notation, it reads:

$$S \to \epsilon \mid (S)S \mid .S$$

With ϵ a terminal (empty) symbol. For the sake of simplicity, these production rules do not include the restriction that the two ends of a base-pair must be distant of at least θ nucleotides. One could modify these rules to produce sequences, as such:

$$S \to \epsilon \mid aSbS \mid aS$$

with a, b with values in $\{\texttt{A}, \texttt{U}, \texttt{G}, \texttt{C}\}$. a and b are required to be compatible in the middle rule. The derivation tree of a sequence yields a structure compatible with the output sequence. Nussinov's algorithm can then be seen as figuring out the *maximum-weight parsing tree* for this grammar and an input sequence. As such, it is a specialization of the CYK algorithm [37, 38] for context-free grammar parsing.

The core idea behind the polynomial algorithms of [32] for RNA FOLDING in weighted base-pairs models is then given in the following proposition. It only covers the pseudoknotted secondary structure case. To include tertiary interactions, the compatibility graph can be augmented with additional gadgets, and the matching generalized to "2-matchings". The reader is referred to [32] for details.

Proposition 1 ([32])**.** RNA FOLDING *is polynomial for weighted base-pair energy models restricted to pseudoknotted secondary structures.*

Proof of Proposition 1. In a secondary structure (Definition 1), each nucleotide is paired-up with at most one other nucleotide. A secondary structure over a sequence S therefore corresponds to a *matching* of the compatibility graph $G_{\mathtt{C}}(S)$. A secondary structure $\mathcal{A}$ minimizing $E_{\mathtt{w\text{-}bps}} = \sum_{(i,j)\in\mathcal{A}} w_{S[i]S[j]}$ can therefore be found by finding a maximum-weight matching on the $G_{\mathtt{C}}(S)$. This can be done in polynomial time, for instance using Edmonds' blossom algorithm [39]. □

By definition, a weighted base-pair energy model misses out on *stacking*. The algorithm of Proposition 1 may therefore output structures with a lot of *isolated* base-pairs, which is considered highly unrealistic. Nonetheless, before looking at the complexity of RNA FOLDING with a more complete energy model, we describe below a strategy to slightly increase the potential of Proposition 1 for producing realistic structures.

Choosing weights. Note that in Proposition 1, the weight of a base-pair (i, j) does not have to be only depending on its nucleotide content $(S[i], S[j])$. It can be *position-dependent*, which allows to potentially include any kind of extra information we might have on nucleotide affinity, beyond the pure free-energy contributions of base-pairing. Examples include *thermodynamic base-pairing probabilities* (see digression about their computation in Box 2) or experimental structure-probing data (such as SHAPE [40, 41]).

Box 2. (Digression: computing base-pair probabilities)
In the thermodynamic equilibrium, the probability of a sequence S adopting a certain structure $\mathcal{A}$ is given by the *Boltzmann distribution*:

$$\mathcal{P}(\mathcal{A}, S) = \frac{1}{Z} e^{-\mathcal{E}(\mathcal{A},S)}$$

where $\mathcal{E}(\mathcal{A}, S)$ is the energy associated to the structure-sequence pair $(\mathcal{A}, S)$. This probability space consisting of structures is also often called the *Boltzmann ensemble*. Given this probability distribution, one may for instance wonder about the probability that a certain base-pair (i, j) occurs, $p_S(i, j) = \sum_{\mathcal{A} \text{ s.t } (i,j)\in\mathcal{A}} \mathcal{P}(\mathcal{A}, S)$

Interestingly, the dynamic programming schemes used for RNA FOLDING, such as Nussinov's (Box 1) and Zuker's algorithm can be modified to compute these base-pair probabilities in polynomial time [20], using a variant of the Inside/Outside algorithm [42]. This foundational result is for instance used in the tool `LoCaRNA` [43], capable of *simultaneous alignment and folding* of a set of input sequences.

Zuker's algorithm. However, the gold standard of RNA energy models is undoubtedly the Turner model. A central algorithmic result regarding RNA FOLDING is therefore Zuker's algorithm [19], which can solve RNA FOLDING for the Turner energy model and conflict-free secondary structures. Several reference implementations exist [24, 44, 26], making it a usual starting point for any RNA structural investigations.

Algorithmic strategy and complexity. Similarly to Nussinov's algorithm, the algorithmic strategy of Zuker's algorithm is dynamic programming over the intervals of the input sequence. As it handles a more complex energy model, it unsurprisingly uses several dynamic programming tables, and simplification hypotheses to keep the complexity low. These hypotheses are described below. As for the details of the dynamic programming equations, the reader is referred to [24].

As mentioned above (Definition 6), the standard simplifications to the Turner energy model involve especially *multi-loops*. Specifically, for a multi-loop $\ell = \{(i_k, j_k)\}_{0 \leq k \leq p}$ ($p > 2$), the energy contribution is simplified to:

$$w(\ell) = a + b \cdot p + c \cdot |j_0 - i_0|$$

Where $|j_0 - i_0|$ takes into account the number of unpaired nucleotides in the loop. With this simplification, Zuker's algorithm uses $O(n^2)$ space and $O(n^4)$ in time. To further reduce the complexity, a standard practice is to impose a constant upper bound onto the length of 2-loops (bulges), yielding $O(n^3)$ time complexity.

Hardness and tractable sub-cases. However, RNA FOLDING does become hard when considering both pseudoknots and realistic energy models [33]. In fact the *stacking* energy model (Definition 4) is enough to prove hardness, with a simple reduction from BIN PACKING.

Given the biological importance [17] of pseudoknots, several approaches have been developed to tackle *tractable sub-cases* of RNA FOLDING with pseudoknots and realistic energy models [16, 29, 30, 45]. All of these approaches are dynamic programming schemes, with sub-problems indexed by sets of positions in the input sequence (and therefore, complicated generalizations of Nussinov's algorithm, Box 1). To make RNA FOLDING tractable, they restrict the exploration to a certain set of pseudoknotted patterns. Each of them was developed with either the motivation of including a specific pseudoknotted pattern into the search space, or expanding this space as much as possible while maintaining a given complexity.

RNA FOLDING within this PhD thesis. Chapter 3 presents a method allowing to specify a finite set of pseudoknotted patterns (e.g, `[{]}` or `[<{][}{][}{][}>]`) and automatically derive a set of dynamic programming solving RNA FOLDING for these patterns. Where the methods mentioned above ([16, 29, 30, 45]) are the result of a tedious hand derivation of dynamic programming equations, the method of Chapter 3 re-derives some of them automatically. In addition, if a pseudoknotted pattern is not covered by an existing method, or covered with prohibitive complexity, it can derive tailored dynamic programming equations to solve RNA FOLDING restricted to this pattern.

1.1.4.2 RNA design

Informally, RNA DESIGN is the problem of producing RNA sequences capable of performing prescribed functions. In the context of this PhD thesis, we are concerned with *structural* RNA DESIGN, namely the task of finding, given a structure, a sequence that preferentially folds according to it. We state it formally below.

Problem 2 (RNA DESIGN).
Input: a set of base-pairs $\mathcal{A}$ from a structure class $\mathcal{C}$
Output: A sequence S such that $\forall \mathcal{A}' \neq \mathcal{A} \in \mathcal{C}, \mathcal{E}(\mathcal{A}, S) < \mathcal{E}(\mathcal{A}', S)$

Where $\mathcal{E}$ is an energy model and $\mathcal{C}$ a structure class. To be precise, this version of RNA DESIGN is also known in the literature as *inverse folding* [46], or *negative* RNA DESIGN, as opposed to *positive* RNA DESIGN [47, 48, 49]. In a nutshell, positive design only asks to find a sequence with an energy smaller than a given threshold for the input structure. It does not require (contrary to our definition above) that *all alternative structures* are less energetically favorable.

Computational hardness. A problem close to RNA DESIGN, namely RNA DESIGN EXTENSION, has been proved NP-hard in [14], with $\mathcal{C}$ restricted to conflict-free secondary structures and $\mathcal{E}$ limited to the Nussinov energy model (Definition 2), i.e. the simplest possible choices. In RNA DESIGN EXTENSION, in addition to a structure $\mathcal{A}$, a *partial assignment* of the sequence is given (i.e., constraints of the form $S[i] = x$ for $x \in \{\texttt{A},\texttt{U},\texttt{G},\texttt{C}\}$), restricting the set of possible outputs. Although this problem is really close to RNA DESIGN (and perhaps closer in spirit to practical design settings), the complexity of RNA DESIGN is strictly speaking still open. It is also interesting to note that this hardness result is surprisingly recent ($\sim$ 2018) for such a central computational biology problem.

Practical relevance and current solutions. Given the versatility of RNA molecules in biological systems, it is not surprising that RNA DESIGN has many potential applications, e.g. in pharmaceutics [2, 50, 51] or synthetic biology [52]. Because of this importance of RNA DESIGN, many (mostly heuristic) tools have been developed to tackle it [53, 54, 49, 55, 44, 48, 56, 57, 58, 59]. One could also cite the online "serious game" `EteRNA` (`https://eternagame.org/`), that crowdsources the resolution of RNA DESIGN instances through gamification.

RNA DESIGN within this PhD thesis. Chapter 2 develops structure-simplifying techniques that we argue could provide a speedup of [48], a sampling-based technique for RNA DESIGN. The idea is that the structure simplification would greatly accelerate the production of sequences, while being compensated by a rejection mechanism to remain correct.

1.1.4.3 Structure-sequence alignment

Informally, the STRUCTURE-SEQUENCE ALIGNMENT problem takes as input an arc-annotated sequence $(Q, \mathcal{A})$ (the "query") and a plain sequence T (the "target"), and asks whether T is a potential *homolog*

of Q. More precisely, it looks for the most parsimonious mutation scenario transforming Q into T while keeping a maximal compatibility with $\mathcal{A}$. Another way of stating it is that the Structure-Sequence Alignment problem is the computation of an *edit distance* between a folded RNA and a plain sequence. In practice, the task is to figure out the best possible mapping between Q and T, where "best" measures structure and sequence conservation. It is formally stated below. Beforehand, we first quickly define a notion of *monotonous* mapping between two sequences.

Definition 8 (monotonous mapping)**.** Given two sequences Q and T, a *monotonous mapping* μ from Q to T is a function $\mu : \{1 \dots |Q|\} \to \{1 \dots |T|\} \cup \{\perp\}$ such that $\forall i, j \in [1 \dots |Q|]$ with $\mu(i) \neq \perp$ and $\mu(j) \neq \perp$, $\mu(i) < \mu(j)$.

In the definition above and the problem statement below, $\mu(i) = \perp$ denotes the *deletion* of nucleotide i in Q. This nucleotide is not mapped anywhere in the target sequence.

Problem 3 (Structure-Sequence Alignment)**.**
Input: Arc-annotated sequence $(Q, \mathcal{A})$, plain sequence T
Output: μ monotonous mapping from Q to T, minimizing:

$$\texttt{score}(\mu) = \sum_{(i,j)\in\mathcal{A}} \gamma_2(i, j, \mu(i), \mu(j)) + \sum_{i \text{ s.t } \mu(i)\neq\perp} \gamma_1(i, \mu(i)) + \sum_{\text{gap } g\subset Q} \lambda_Q|g| + c_Q + \sum_{\text{gap } g\subset T} \lambda_T|g| + c_T$$

Where:

- γ_2 measures *structure conservation*. It rewards the mapping of the ends of a base-pair onto compatible nucleotides, and penalizes deletion (one of the ends has value $\perp$) or mapping onto incompatible nucleotides.
- γ_1 measures *sequence conservation*. It rewards the mapping of a nucleotide onto the same nucleotide. Note that it does not handle the deletion case ($\perp$), which is taken into account by the *gap* terms below.
- A gap g in the query, informally denoted by $g \subset Q$ in the formula above, is a set of consecutive positions that are all deleted, i.e. mapped to $\perp$. An *affine* cost is associated to this gap, with parameters λ_Q and c_Q. $|g|$ is the number of consecutive deleted positions.
- A gap g in the target is made up of the positions in $T[\mu(i-1)+1 \dots \mu(i+1)-1]$ for two consecutive positions $i, i+1$ of the query, such that $\mu(i) \neq \perp$, $\mu(i+1) \neq \perp$ and $\mu(i+1) > \mu(i)+1$. As for gaps in query, an affine cost is also associated to this gap, with parameters λ_T and c_T.

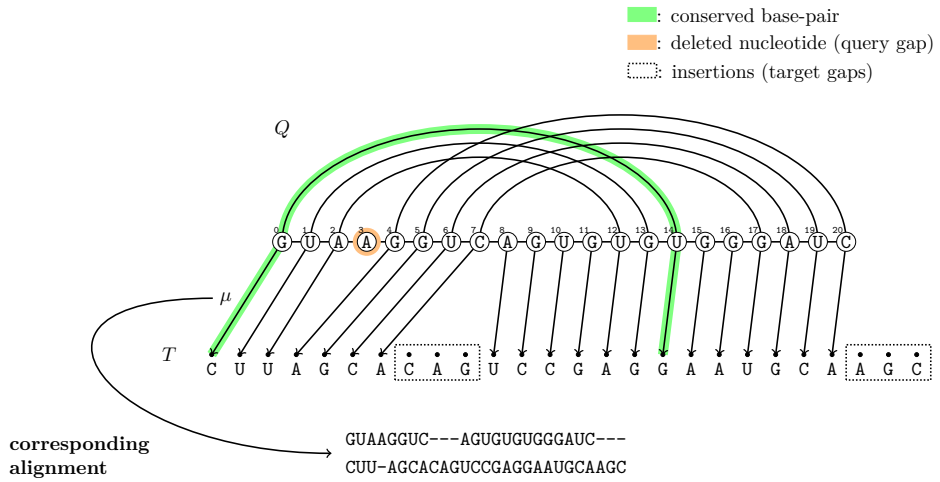


Figure 1.8: Illustration of the definition of Structure-Sequence Alignment. An arc-annotated sequence Q is mapped onto a plain sequence T by an application μ. The task is to find the best mapping μ according to a scoring scheme that takes into account both structure and sequence conservation. Gaps in the query (deletions) and in the target (insertions) are also penalized.

The definition of Structure-Sequence Alignment is illustrated on Figure 1.8. Note that an alignment of Q and T can be built from μ in a straightforward way: deletions (i such that $\mu(i) = \perp$) induce gap symbols in the target sequence, while insertions ($\mu(i+1) > \mu(i)+1$) induce them in the query.

Computational hardness. Table 1.2 sums up important hardness and algorithmic results for Structure-Sequence Alignment. As for RNA folding, including pseudoknots into the picture makes the problem intractable. Note that [60], where hardness was proven, states the problem as the computation of an *edit distance* between RNA structures, which might be of independent interest to the reader.

Practical importance, current solutions, and their limits. Structure-Sequence Alignment is the computational workhorse underlying *homology search*, that is to say the search for occurrences of a given structured RNA in sequence databases. The leading tool for such a task in the absence of pseudoknots is `InfeRNAl` [61], which represents queries as *covariance models* (defined in Chapter 4). It is the tool at the core of `Rfam` [11], which regroups detected homologs into families. Because `InfeRNAl` cannot take pseudoknots into account, pseudoknotted families in `Rfam` have to be partially curated manually. When expanding them, the crucial homology search step does not take pseudoknots into account, which may lead to both missed hits and false positive hits. The overall result is an underestimation of the prevalence and importance of pseudoknotted functional RNAs. Algorithms for pseudoknotted Structure-Sequence Alignment do exist [62], but are currently too computationally expensive to be used in practice.

energy model $\mathcal{E}$ ↓ \ structure class $\mathcal{C}$ →	STRUCTURE-SEQUENCE ALIGNMENT	conflict-free	PK sec	tertiary
	Nussinov (Def. 3)	**P** [61, 62]	**NP-hard** [60]	
	Turner (Def. 6)			

Table 1.2: Like RNA FOLDING, STRUCTURE-SEQUENCE ALIGNMENT becomes hard when pseudoknots are included into the picture.

STRUCTURE-SEQUENCE ALIGNMENT **within this PhD thesis.** The method presented in Chapter 2 suggests a *hierarchical approach* to pseudoknotted homolog search. It allows indeed to simplify an input arc-annotated sequence while losing a minimum amount of information. These simplified models can then be used to quickly filter out uninteresting sequences, so that the full (expensive) model is only run to refine results. Then, in Chapter 4, we show an empirical evaluation of `LiCoRNA` (`https://licorna.lri.fr/`, implementation of [62]) for pseudoknotted STRUCTURE-SEQUENCE ALIGNMENT, and suggest a generalization of covariance models to the pseudoknotted case.

1.1.4.4 RNA barrier

Compared to the problems defined above, the RNA ENERGY BARRIER problem takes a more dynamic view of RNA structures. Indeed, given two possible folded structures for a sequence, it asks whether there is a feasible spontaneous *transition* between the two. A spontaneous transition is one that would occur from thermal fluctuations only. In physical terms, the question is whether the two input structures are separated by an unreachable *energy barrier* or not. From Arrhenius' law, the energy barrier is indeed what dictates the *transition rate* between two conformations, with an exponential dependence. With $\texttt{barrier}(\mathcal{A}_1 \to \mathcal{A}_2)$ the energy barrier between two structures $\mathcal{A}_1$ and $\mathcal{A}_2$, it reads:

$$P(\mathcal{A}_1 \to \mathcal{A}_2) \propto e^{\frac{-\texttt{barrier}(\mathcal{A}_1 \to \mathcal{A}_2)}{RT}}$$

With R a constant and T the temperature.

Problem statement. In the statement below, structures are taken as sets of base-pairs. Therefore, for two structures A and B, $A \Delta B = 1$ (symmetric difference) means that B is obtained from A by either adding or removing a single base-pair.

Problem 4 (RNA ENERGY BARRIER).
Input: Two structures L and R from a class $\mathcal{C}$, a sequence S, and a threshold k
Output: If it exists, a sequence of structures $A_0 = L, \ldots, A_p = R$ in $\mathcal{C}$ such that $\forall i \in [0 \ldots p-1]$, $A_{i+1} \Delta A_i = 1$ and $\forall i \in [0 \ldots p]$, $\mathcal{E}(A_i) \leq \mathcal{E}(L) + k$.

Note that the energy upper bound is given with respect to the starting point. An example of instance and solution to this problem is given on Figure 1.9 (left).

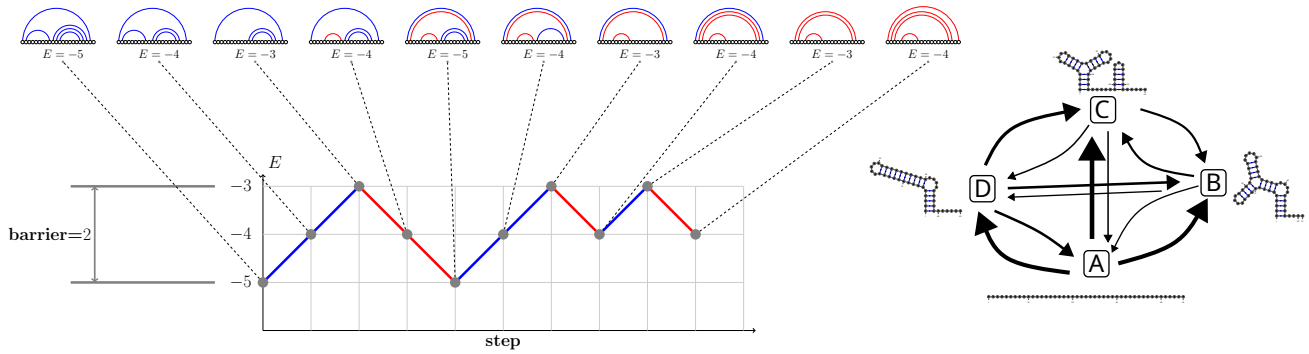


Figure 1.9: (left) Example of an instance of RNA Energy Barrier, and a possible optimal schedule. The barrier is here equal to 2, measured as the energy difference between the starting point and the culminating energy along the reconfiguration. (right) The ultimate purpose of computing barriers is often to get a quantitative idea of the conformational landscape of an RNA sequence. In this picture, the barrier influences the width of the arrows (easiness of transition) between structures. The lower the barrier, the easier a transition is.

Computational hardness. The RNA Energy Barrier problem is already NP-hard for conflict-free secondary structures as $\mathcal{C}$ and the Nussinov energy model (Definition 2) [63], i.e. the simplest possible choices.

Practical importance. RNA molecules are synthesized linearly by the RNA polymerase, i.e. nucleotide by nucleotide in a given order (see Figure 1.1). It is therefore reasonable to believe that RNA molecules are either initially synthesized with an empty "all-open" structure, or folded as a result of progressive "co-transcriptional" folding [64]. Given that RNA molecules also have a limited life-span, the set of structures they adopt is necessarily limited to a landscape region reachable with *a low energy barrier* starting from this initial fold. One of the ultimate goal is to be able to predict, as a function of time, the continuous change in probability of adopting the different structures in that region [65] (or, equivalently, *concentrations* for these structures as a function of time).

Current approaches. Given the practical importance of RNA Energy Barrier, it is not surprising that many frameworks [66] and heuristic tools [67, 68] have been developed to get a grasp of RNA energy landscapes. Many of them rely on an exploration of sets of *suboptimal* structures [69, 70], i.e. having low energy, but not as low as the minimum-free energy (MFE) structure. Once important points in the conformational landscape have been identified, the energy barrier is what determines the amount of *transitioning* between the points, as illustrated on Figure 1.9 (right).

RNA Energy Barrier **within this PhD thesis.** Chapter 5 connects the RNA Energy Barrier problem with *independent set reconfiguration* on bipartite graphs and the problem of computing the *directed pathwidth* of an associated graph (these notions are defined in Chapter 5). Given these connections, it formulates *parameterized algorithms* for it (section 1.2, below).

1.2 Parameterized algorithmics

The originality of this PhD thesis is that it tackles the RNA bioinformatics problems defined above with techniques from *parameterized algorithmics* [71], a relatively recent and very dynamic field of algorithm research. Its philosophy and basic definitions are given in sections 1.2.1 and 1.2.2.

A specific kind of parameterized algorithmics based on *width parameters* (especially treewidth) was used in this PhD thesis. Section 1.2.3 will introduce this notion.

1.2.1 Philosophy and basic definitions

Parameterized complexity and algorithms: philosophy. When dealing with an NP-hard problem, it is often the case that it contains tractable sub-cases. For instance, MaxCut is NP-hard in general but polynomial on planar graphs. Parameterized algorithmics generalizes this idea, by formulating algorithms whose complexities depend on multiple *parameters* such that, when the parameter(s) have low value(s), the algorithm is tractable.

But it does more than that: it distinguishes between several kinds of behaviors with respect to the parameter, and provides a hardness theory to decide between them. To state these concepts more precisely, let us start with the following definition, taken from [71]:

Definition 9 (Parameterized problem)**.** A parameterized problem is a language $L \subseteq \Sigma^* \times \mathbb{N}$. For an instance $(x, k) \in \Sigma^* \times \mathbb{N}$, k is called the *parameter*.

For instance, x might be a graph, and k an integer characteristic computable from the graph (its maximum degree, the number of times a specific pattern appears...). In other cases, the parameter k could be part of the *problem specification*, such as the number of colors in a Coloring problem, or the number of parts to divide the graph into in a partitioning problem. A classic example of a parameter is also the *size of the output solution*, such as k when wondering if the graph contains a k-clique. The question parameterized algorithmics asks is then whether one can design algorithms whose run-time is low when the parameter is low. More precisely, we want the complexity of the algorithms to be polynomial when k is *fixed*. The definitions below distinguish two cases for such a complexity, depending on whether the *degree* of the polynomial depends on k.

Polynomial algorithms for fixed parameter: XP and FPT. An algorithm with a complexity that is polynomial for a fixed value of k, but for which the degree of the polynomial depends on k is said to be XP:

Definition 10 (slicewise-polynomial: XP)**.** An algorithm is said to run in time *XP for parameter* k if its complexity has an upper bound of the form $f(k) \cdot n^{g(k)}$ for f, g two computable functions.

The name "slice-wise polynomial" stems from the fact that for each "slice" of problem corresponding to a given value of k, the algorithm complexity is a different polynomial. When the polynomial degree is the same for all parameter values, we are in the much more favorable FPT case:

Definition 11 (fixed-parameter tractability: FPT)**.** An algorithm is said to run in time *FPT for parameter* k if its complexity has an upper bound of the form $f(k) \cdot n^c$, for f a computable function.

To illustrate these definitions, we will consider the CLIQUE problem, defined below, as a running example.

Problem 5 (Clique)**.**
Input: graph G, integer k
Question: Does G contain a k-clique?

In Box 3, we give two parameterizations for Clique, one yielding an FPT algorithm (maximum degree Δ of the input graph) and one yielding an XP algorithm (k). The next section will present a hardness theory (the W-hierarchy) allowing to provide evidence that an FPT algorithm for a given parameterized problem is unlikely to exist.

Box 3. (Parameterizations for Clique)
Consider the following algorithm for Clique (from [71]), with Δ the maximum degree of the input graph:

1. If $\Delta < k - 1$ output no.
2. Else, for each vertex v, test whether it is part of a k-clique by checking if its neighborhood contains a k-clique. To do so, check all $O(2^{\Delta})$ subsets of $N(v)$, and check in $O(\Delta^2)$ whether it is a clique or not.

The overall run-time of the algorithm is $O(2^{\Delta} \cdot \Delta^2 \cdot n)$. It is therefore an FPT algorithm for Clique parameterized by Δ.

Another algorithm for Clique is a straightforward iteration over all k subsets in $O(n^k)$, along with a $O(k^2)$ test for each of them to see if it is a clique. This constitutes an XP algorithm for Clique. As we shall see in Section 1.2.2, Clique is in fact W[1]-hard with k as a parameter, which is considered convincing proof that it does not allow for an FPT algorithm with this parameterization.

1.2.2 Parameterized intractability

Parameterized complexity comes with a hardness theory to rule out FPT or XP algorithms when studying a parameterized problem. We start this section by presenting Para-NP-hardness, which rules out XP and FPT. Then, we move on to W[1]-hardness, which allows distinguishing between XP and FPT.

Excluding XP and FPT: Para-NP-hardness.

Definition 12 (Para-NP-hardness)**.** A parameterized problem is *Para-NP-hard* if it is NP-hard for a fixed value of the parameter.

For example, consider the Coloring problem, which takes as input a graph G and an integer k, and asks whether G admits a proper coloring with less than k colors. That is to say, a coloring assigning different colors to connected vertices. It is already NP-hard to decide whether a graph G admits a proper 3-coloring. Coloring parameterized by the number of colors is therefore Para-NP-hard. The existence of an XP algorithm for coloring parameterized by k, i.e. running in time bounded by $f(k) \cdot n^{g(k)}$, is therefore ruled out, unless P=NP.

Parameterized reductions. Just like in NP-hardness theory, reductions between problems allow distinguishing polynomial from NP-hard problems, it is here a notion of *parameterized reduction* that is at the basis of distinguishing FPT from XP. The definition is given below. It is such that it transfers fixed-parameter tractability: if there is a parameterized reduction from A to B and B is FPT, then so is A.

Definition 13 (parameterized reduction)**.** Let $A, B \subseteq \Sigma^* \times \mathbb{N}$ be two parameterized problems. A *parameterized reduction* from A to B is an algorithm $\mathcal{A}$ that, given an instance (x, k) of A, outputs an instance (x', k') of B, such that:

1. (x, k) yes-instance $\Leftrightarrow$ (x', k') yes-instance.
2. $k' \leq g(k)$ for some computable function g
3. $\mathcal{A}$ runs in FPT time with respect to k

Conversely, if we have good reasons to believe that there is no FPT algorithm for A, then this impossibility is transferred to B. The role played by the class NP in NP-hardness theory is here taken up by a hierarchy of classes of parameterized problems called the W-hierarchy, which we now present.

Ruling out FPT: W-hierarchy. The W-hierarchy is a hierarchy of computational classes indexed by an integer t, $\{W[t] \mid \forall t > 0\}$, with $W[0] = \text{FPT}$. Its precise definition, in terms of Weighted Circuit Satisfiability, is beyond the scope of this manuscript. Details can be found in [71]. The classes of most interest to the algorithm designer are $W[1]$ and $W[2]$, for which there are natural complete problems. The fundamental hypothesis underlying parameterized complexity is then:

$$\text{FPT} \neq W[1]$$

As other fundamental working hypotheses in computational complexity, it is not formally proven. It is widely believed to be true, in part because of the following result, which is the starting point of a lot of parameterized reductions:

Theorem 1 ([71])**.** Clique *parameterized by the solution size is* $W[1]$*-complete.*

The assumption that FPT $\neq W[1]$ is therefore also supported by the fact that we do not know of any $n^{o(k)}$ algorithm for Clique (with k the solution size) despite decades of research involving this fundamental problem.

Parameterized reductions within this PhD thesis. All chapters of this PhD thesis except Chapter 3 involve Para-NP-hardness proofs and parameterized reductions. Some of them use restricted versions, which are still $W[1]$-complete, namely Multi-Colored Clique and Clique on regular graphs, which we briefly introduce before closing this section.

Multi-Colored Clique consists in asking, given a partition $(X_1, \ldots, X_k)$ of the vertices of the graph as input, whether it contains a k-clique with one vertex in each X_i.

Problem 6 (Multi-Colored Clique)**.**
Input: graph G, integer k, partition $(X_1, \ldots, X_k)$ of V
Question: Does G allow for a k-clique containing exactly one vertex in each X_i, for $1 \leq i \leq k$?

We refer below as "multicolored clique" to a k-clique having one vertex in each part X_i of a partition. As for regular graphs, they simply are graphs in which all vertices have the same degree. The combination of both, Multi-Colored Clique on regular graphs is still $W[1]$-complete parameterized by the solution size. Parameterized reductions from Clique to these restricted versions are given in Box 4. These proofs can also be found in [71], but we include them here for self-completeness and illustration purposes.

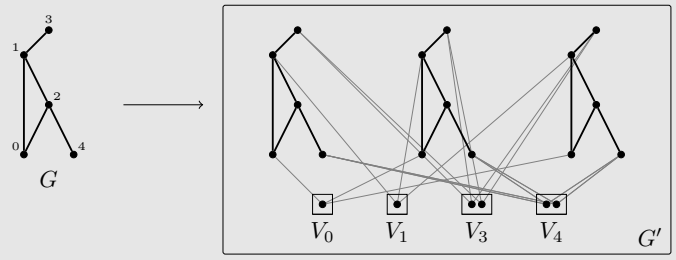
1.2.3 Width parameters: the example of treewidth

So far, the examples of parameterizations we have seen are the solution size (k when looking for a k-clique) or the maximum degree of a graph. However, this PhD thesis focuses on the application to structural RNA bioinformatics of another kind of parameters, namely *graph widths*. This section provides a primer of *structural graph theory*, from which graph widths stem. It focuses mainly on *treewidth*, the most studied and well understood of these parameters, and also the one that is mostly used in this PhD thesis.

Structural graph theory: philosophy. When confronted with an NP-hard problem on graphs in practice, it is of paramount importance to investigate whether we are solving the problem in its full generality, or if we are in fact only looking at instances with a specific structure that could be exploited algorithmically. The goal of structural graph theory is to describe and characterize such potential structural restrictions [72]. A particularly useful idea in that context is that of *separation*, i.e. the possibility of breaking down graphs into small pieces, which may be handled separately.

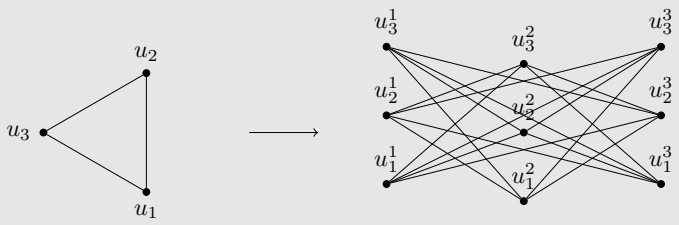
Graph decompositions, such as *tree decompositions* (defined below) are data structures describing these small pieces and how they connect together. The associated *width* (an integer) is intuitively

Box 4. (Multi-Colored Clique on regular graphs)
Let us start with the reduction from Clique to Clique on regular graphs. Given a graph $G = (V, E)$ and an integer k, the purpose is to build a regular graph G' and an integer k' such that G has a k-clique if and only if G' has a k' clique. Recall that a regular graph is one in which all vertices have the same degree. To start with, note that the case $k \leq 2$ (does G have an edge?) is trivial for Clique. We therefore restrict our attention to $k \geq 3$.
Let us note d the maximum degree of G, and $d(u)$ the degree of a vertex $u \in G$. G' consists of d copies of G, as well as vertex sets V_u, for each $u \in V$. $|V_u| = d - d(u)$, and each copy of the d copies of u are connected to each vertex in V_u. This construction is illustrated on the drawing below for a simple graph.



We argue that G' has a k clique if and only if G has a k clique. The "if" part is straightforward as G' contains copies of G. For the "only if" part, note that the vertices in the sets $\{V_u\}_{u \in V}$ are not involved in any triangle, and therefore in no k-clique (for $k \geq 3$). The cliques of G' are therefore only the cliques in the copies of G. For $k' = k$, we do have the equivalence of k-clique containment in both graphs. G' being of $O(|V|^2)$ size, the reduction is polynomial, and all conditions of a parameterized reduction are met.

From Clique on regular graphs to Multi-Colored Clique. We now formulate a parameterized reduction from Clique on regular graphs to Multi-Colored Clique on regular graph, both parameterized by k. Consider therefore a regular graph $G = (V, E)$, and call r the common degree value of all of its vertices. The purpose is to build $G' = (V', E')$, and give a partition $(X_1, \ldots, X_k)$ of V', such that G contains a k-clique if and only if G contains a multicolored clique. Let us build G' as follows. V' is composed of k sets $V_1, \ldots, V_k$ of cardinality $|V|$. We further denote V as $\{u_1, \ldots, u_n\}$ and each V_i as $\{u_1^i, \ldots, u_n^i\}$. Now, for $i \neq j$, $(u_k^i, u_l^j) \in E'$ if and only if $(u_k, u_l) \in E$. Crucially, no edges internal to any V_i are added. We illustrate the transformation below.



Now, given a k-clique $\{u_i\}_{i \in I}$ with $I = (i_1, \ldots, i_k)$ in G, a multicolored k-clique in G' is simply $\{u_{i_j}^j\}_{1 \leq j \leq k}$. Conversely, a multicolored k-clique in G', $\{u_{i_j}^j\}_{1 \leq j \leq k}$ yields a clique $\{u_{i_j}\}_{1 \leq j \leq k}$ in G.

either the size of these pieces, or that of their *interfaces*. Let us now put these concepts into more precise terms through the definition of treewidth and tree decompositions.

Treewidth and tree decompositions: definitions. Treewidth is an integer quantifying the *tree-likeness* of a graph. It is the smallest possible *width* of a tree decomposition of a graph, which is in short a "tree of bags of vertices" with certain properties:

Definition 14 (Tree decomposition)**.** Given a graph $G = (V, E)$, a tree decomposition $\mathcal{T}$ is a tuple $(T, \{X_t\}_{t \in T})$ with T a tree, and $\forall t \in T$, X_t a set of vertices of the graph, such that:

1. $\forall u \in V$, the set $T_u = \{t \in T \mid u \in X_t\}$ must be a connected sub-tree of T.
2. $\forall (u, v) \in E$, T_u and T_v must intersect.

Notation 2 (vertex subtree)**.** $T_u = \{t \in T \mid u \in X_t\}$

The width $w(\mathcal{T})$ of a tree decomposition $\mathcal{T}$ of G is $\max_{t \in T} |X_t| - 1$, the size of its biggest bag minus one. The treewidth $tw(G)$ of a graph G is the minimum possible width of a tree decomposition of G. The "minus one" in the definition of the width of a decomposition is there so that trees have a treewidth of 1. Examples of graphs along with optimal tree decompositions are given on Figure 1.10.

Tree decompositions and separators. To better understand why the definition above is a good way to characterize "tree-like" graphs, we formulate and prove here a basic result on the separating properties of tree decomposition. To simplify the exposition, we state below the essence of the result, and present the formal details in Box 5.

Property 1. *The intersections of adjacent bags in a tree decomposition of a graph G are* separators *of G.*

Where a *separator* is a set of vertices that disconnects the graph when removed. Property 1 gives us an intuitive reason why tree decompositions can be useful algorithmically. Given an edge (i, j) of the tree decomposition with $S = X_i \cap X_j$ a separator of G, we can indeed imagine solving a problem on each component of $G \setminus S$, and combine the result through the interface $S = X_i \cap X_j$ of size bounded by $tw(G)$.

Dynamic programming on tree decompositions. Thanks to the separating properties defined above, an extremely wide variety of combinatorial optimization problems can be solved efficiently on graphs of bounded treewidth. This variety has been made particularly apparent by the following (much celebrated) theorem. It is stated here informally, the interested reader can find more details in [71]

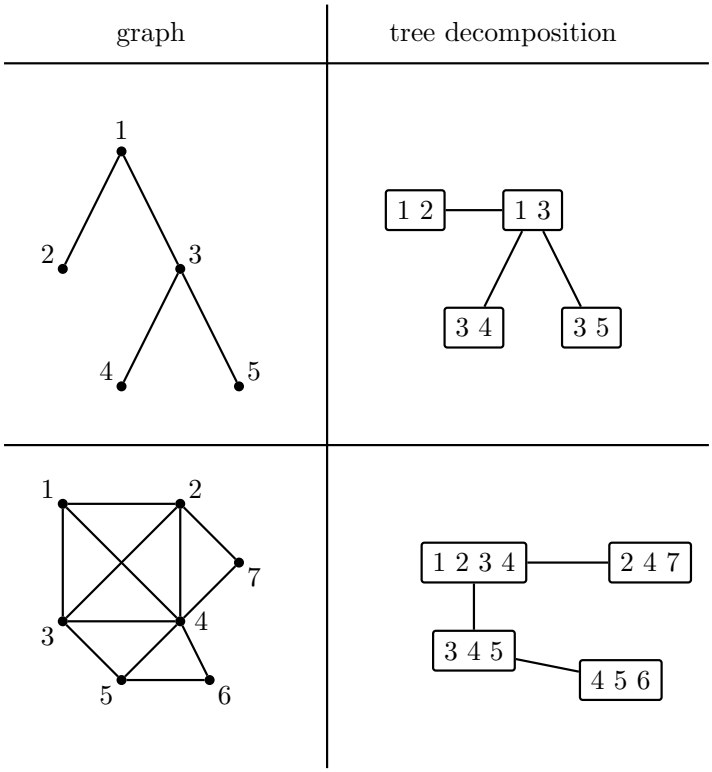


Figure 1.10: Examples of graphs and possible optimal tree decompositions for them. (top) A tree has treewidth 1, i.e. allows for a tree decomposition with bags of size 2. They correspond to its edges, connected following the tree structure. (bottom) A slightly more complicated graph, of treewidth 3. The fact that it contains a clique of size 4 gives us a lower bound of 3 on its treewidth, hence this tree decomposition is optimal.

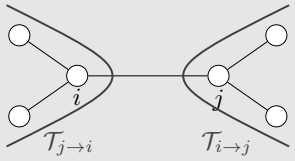
Box 5. (Adjacent bag intersections are separators)

Let us start with a formal definition for a separator in a graph G.

Definition 15 ((a, b)-separator)**.** Given a graph $G = (V, E)$, $S \subseteq V$, and $a, b \in V$, S is an (a, b)-separator if a and b are in two different connected components of $G \setminus S$.

Given this definition, S is said to be a *separator* of G if $\exists a, b \in V$ such that S is an (a, b)-separator.

Much like any (non-leaf) vertex of a tree separates it in several parts when removed, we prove now a very useful basic result: the intersections of the bags of a tree decomposition are, in the general case, separators of the graph. To state the result, we introduce the following notation: given an edge (i, j) of T, we write $\mathcal{T}_{i\to j}$ and $\mathcal{T}_{j\to i}$ the two sub-trees obtained when removing edge (i, j). $\mathcal{T}_{i\to j}$ is the one containing j, and $\mathcal{T}_{j\to i}$ contains i. This definition is illustrated on the figure below.



We then write $V(\mathcal{T}_{i\to j})$ and $V(\mathcal{T}_{j\to i})$ for the unions of the bags corresponding to these subtrees (e.g. $V(\mathcal{T}_{i\to j}) = \bigcup_{t\in\mathcal{T}_{i\to j}} X_t$). With these notations, we have:

Property 2 (bag intersections are separators)**.** *Given $\mathcal{T} = (T, \{X_t\}_{t\in T})$ a tree decomposition of a graph G and (i, j) an edge of T, if $V(\mathcal{T}_{i\to j}) \setminus V(\mathcal{T}_{j\to i}) \neq \emptyset$ and $V(\mathcal{T}_{j\to i}) \setminus V(\mathcal{T}_{i\to j}) \neq \emptyset$ then $X_i \cap X_j$ is a separator of G.*

Proof. Let $a \in V(\mathcal{T}_{i\to j}) \setminus V(\mathcal{T}_{j\to i})$ and $b \in V(\mathcal{T}_{j\to i}) \setminus V(\mathcal{T}_{i\to j})$. By definition of a tree decomposition, T_a and T_b (Notation 3) are connected sub-trees of $\mathcal{T}$, and therefore $a \notin X_i \cap X_j$ and $b \notin X_i \cap X_j$. If a and b are not in the same connected component in G, then they are not in the same components of $G \setminus (X_i \cap X_j)$ and we are done.
Otherwise, let $P = (s_0, \ldots, s_p)$, with $s_0 = a$ and $s_p = b$, be any path between a and b. We show that P necessarily intersects $X_i \cap X_j$. Let t be the largest integer in $[0 \ldots p]$ such that $s_t \in V(\mathcal{T}_{i\to j}) \setminus V(\mathcal{T}_{j\to i})$. By definition of a, b and P, $0 \leq t < p$. Therefore $s_{t+1} \in V(\mathcal{T}_{j\to i})$. But since (s_t, s_{t+1}) is an edge of G, their two sub-trees T_{s_t} and $T_{s_{t+1}}$ must intersect. s_{t+1} must therefore intersect be present in both X_i and X_j to "reach" towards s_t.
Thus, any path P between a and b must intersect $X_i \cap X_j$. When removing it, all paths are cut, and since $a \notin X_i$ and $b \notin X_j$, a and b are indeed in two different components of $G \setminus (X_i \cap X_j)$. $X_i \cap X_j$ is a separator of G.

□

Theorem 2 (Courcelle's theorem [73]). *Any graph problem expressible in Monadic Second Order Logic (MSOL) is* FPT *by the treewidth of the input graph.*

This encompasses pretty much any classic graph problem. However, due to the impractical complexity it typically yields (with towers of exponential [74]), this theorem should only be regarded as a theoretical classification tool. Practical FPT algorithms parameterized by treewidth are typically rather designed by hand, and consist in dynamic programming over (rooted) tree decompositions. Box 6 gives an example of such an algorithm, for Maximum Independent Set. It applies the typical strategy of "considering all possible solution assignments" to the intersection between two bags, and then merging sub-solutions given an assignment.

Computing treewidth. Algorithms such as the one presented in Box 6 work with a tree decomposition of the input graph G. How to compute such a tree decomposition must therefore be addressed, and the complexity of doing so taken into account. This is especially important considering the hardness of computing treewidth exactly:

Theorem 3 ([76]). *Given a graph G and an integer k, it is* NP*-hard to decide whether $tw(G) \leq k$.*

However, when the purpose is to show that a given problem is FPT by treewidth, we are usually saved by the following result.

Theorem 4 ([77]). *Given a graph G and an integer k, deciding whether $tw(G) \leq k$, and outputting a tree decomposition of width $\leq k$ if the answer is yes, can be done in $k^{O(k^3)} \cdot n$*

For instance, the algorithm of Box 6 solves MIS in a time FPT in the width of a *given* a tree decomposition. This is not exactly the same as being given only the graph, without a tree decomposition. Thanks to Theorem 4, we know that adding the computation of an optimal tree decomposition to the complexity does not change the fixed-parameter tractability. To avoid the double exponential of $k^{O(k^3)}$, one could also use a constant-factor approximation to treewidth with single-exponential ($2^{O(k)}n^{O(1)}$) FPT complexity [78, 79].

Note that the landmark result of Theorem 4 ([77]) was very recently improved to $k^{O(k^2)}n^{O(1)}$ [80].

Practical treewidth computation. In practice though, when implementing treewidth-based approaches, we typically use one of the numerous existing heuristics [81]. Several optimized exact solvers, such as [82], also exist. Most are an output of the PACE 2017 challenge on treewidth solving [83].

Obstacles to having small treewidth. To gain a better intuition of the limitations imposed on graphs by bounding treewidth, we describe here some useful tools when looking for *treewidth lower bounds*. This will become handy in Chapters 2 and 3 of this thesis. Let us start with the clique.

Box 6. (An FPT algorithm by $tw(G)$ for Maximum Independent Set)
Let us first recall the definition of Maximum Independent Set:

Problem 7 (Maximum Independent Set)**.**
Input: a graph $G = (V, E)$, an integer k
Question: Is there $S \subseteq V$ such that $|S| \geq k$ and $\forall u, v \in S, (u, v) \notin E$?

To obtain a better complexity, the algorithm below relies on the notion of *nice tree decomposition* [75]. It typically greatly simplifies the formulation of such algorithms, even reducing their complexity.

Definition 16 (nice tree decomposition)**.** A *nice* tree decomposition $\mathcal{T} = (T, \{X_t\}_{t \in T})$ of a graph $G = (V, E)$ contains an empty root R, and 3 kinds of bags:

- **"introduce"**: X with exactly one child Y such that $X = Y \cup \{u\}$ for some $u \in V$.
- **"forget"**: a bag X with exactly one child Y such that $Y = X \cup \{u\}$ for some $u \in V$.
- **"join"**: a bag X with exactly two children Y_1, Y_2 with the same content.

Any tree decomposition $\mathcal{T}$ of a graph G can be turned *nice* in linear time [75]. A set S containing no internal edges is called an *independent set* (IS). Imagine now that you are given a *nice* tree decomposition of G of width w (see the paragraph below about "computing treewidth"), and designate any random bag as its root. Each non-root bag X_i in the tree decomposition now has a well-defined parent P, and children $Y_1, \ldots, Y_\ell$. We also call $\mathcal{T}_i$ the sub-tree-decomposition rooted at X_i, and $V_i = V(\mathcal{T}_i)$ the vertices it contains. We can now define the following dynamic programming table, with $S \subseteq X_i \cap P$:

$$c[X_i, S] = \text{maximum size of an IS of } G[V_i] \text{ containing } S$$

If S is an IS, and $c[X_i, S] = -\infty$ otherwise. Recursive relations for c depend on whether X is an "introduce", "forget", or "join" node. Taking the notations of Definition 16 let us start with the join node:

$$c[X, S] = c[Y_1, S] + c[Y_2, S] - |S|$$

Where the $-|S|$ avoid double counting of the cardinality of S. For a "forget" node (i.e., such that $Y = X \cup \{u\}$):

$$c[X, S] = \max(c[Y, S \cup \{u\}], c[Y, S])$$

The two cases correspond to deciding whether u is part of the candidate solution or not. Last, for an introduce node ($X = Y \cup \{u\}$):

$$c[X, S] = c[Y, S \cap Y] + |S \cap \{u\}|$$

As $\forall X_i, |X_i \cap P| = |S| \leq w$, this table has $O(n \cdot 2^w)$ entries. Since the recursive equations always only involve a constant number of cases, we get the same complexity for the algorithm run-time, which is FPT in w. Combined with the fact that computing the treewidth of G is FPT by $tw(G)$ (Theorem 4), we get the result. Full proofs for the recursive equations can be found in [71] and [75].

Property 3. *If G is a graph, $\mathcal{T}$ a tree decomposition of G, and C a clique of G, then there is a bag of $\mathcal{T}$ containing C.*

Proof. The proof is by induction on $k = |C|$, starting at 2 (edges). Supposing the property is true for cliques of size $k-1$, consider $G, C, \mathcal{T}$ as above such that $|C| = k$. Suppose no bag of $\mathcal{T}$ contains C entirely, and pick $v \in C$. $|C \setminus \{v\}| = k-1$ is a clique, and must be contained in some bag X, by induction hypothesis. Let P be the shortest path in $\mathcal{T}$ towards the closest bag containing $\{v\}$, that we denote Y. All edges of C must be represented in $\mathcal{T}$, so all vertices in $C \setminus \{v\}$ must all be present in P, up to Y included. Y therefore contains C. □

But we can do better than this. Treewidth is indeed fundamentally connected to the notion of *minor*:

Definition 17 (minor)**.** A graph H is a *minor* of a graph G if it can be obtained from G through a sequence of vertex deletions, edge deletions, and edge contractions.

A nice introduction to graph minor theory is [84]. The basic property connecting minors and treewidth is then the following:

Property 4. *If H is a minor of G, then $tw(H) \leq tw(G)$*

Proof. Let us handle the three cases of vertex deletion, edge deletion and edge contraction, given a tree decomposition $\mathcal{T}$ of G:

- **vertex deletion** of u: delete any occurrence of u in $\mathcal{T}$
- **edge deletion**: no modification needed.
- **edge contraction** of u into v: replace all occurrences of u by occurrences of v.

The result is a valid tree decomposition of H of lower width. □

Finding cliques as minors, or any other graph of large treewidth (such as a grid) can therefore also be useful to find lower bounds to the treewidth of a graph. It also implies that the class of graphs of treewidth smaller than a given value is *minor closed*. By the Robertson-Seymour theorem [85], it implies that having bounded treewidth is characterized by a finite set of *excluded minors*. Treewidth plays actually a central role in the 20-paper long proof of this theorem, the so-called *graph minor project*. This characterization of bounded treewidth in terms of a finite set of excluded minors will be used in Chapter 2, along with the fact that testing whether a graph contains a fixed minor or not can be done in polynomial time [86][3].

[3] The reader might wonder why this does not help how to decide $tw(G) \leq k$ in polynomial-time, the catch is the evolution of the *number* of minors to test as a function of k, which grows very quickly.

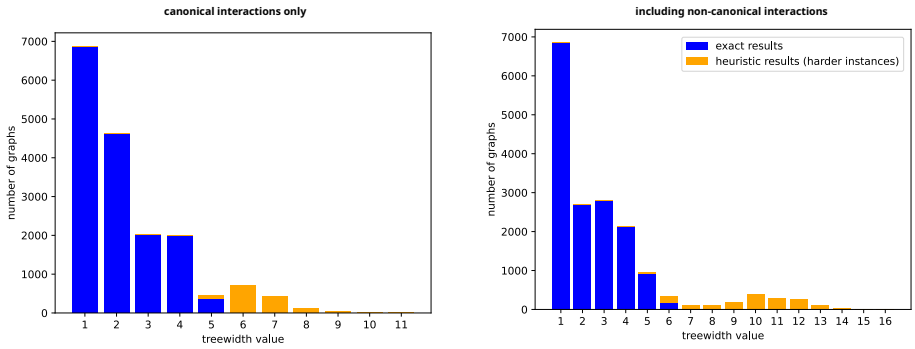


Figure 1.11: Histogram of treewidth values over all RNA-only structures in the PDB database [88]. The data consists of 5 760 non-redundant graphs, each corresponding to a "chain" of a PDB entity. The nucleotide chains and their base pairs were extracted using the DSSR tool [22]. Empty structures (treewidth=1, no base-pairs) have been removed. On each of these graphs, 4 standard treewidth heuristics from the LibTW library [89] (min-degree, min-fill-in, lex-BFS, max-cardinality-search) were launched, as well as an exact solver [82]. A time-out of one minute was given to the exact solver. Results in blue correspond to graphs for which an exact result was found within the time-out. Results in yellow are graphs for which the exact solver exceeded the time-out, and for which the best width result out of the 4 heuristics was taken. Non-canonical interactions are non Watson-Crick or `G-U` connections between nucleotides [5]. When taking them into account, a nucleotide may be involved in more than one base-pair.

Treewidth in RNA bioinformatics. Is treewidth a good parameterization for structural RNA bioinformatics? Figure 1.11 shows a histogram of treewidth values for RNA structures in the PDB database. The graph corresponding to an RNA structure simply consists in a vertex per nucleotide, and an edge for each base-pair and backbone link. Given that these graphs can contain thousands of vertices, these treewidth values are quite reasonable. It is therefore not surprising that there has been attempts to exploit it algorithmically. Results in that respect include [62, 87] (for Structure-Sequence Alignment) and [48] (for RNA design).

Treewidth and tree decompositions in this PhD thesis. Treewidth and tree decompositions constitute one of the central concepts in this PhD thesis. Chapter 2 presents results related to the reduction of a treewidth of a graph through the deletion of a minimal number of edges. The purpose was to unlock the full potential of the algorithms mentioned above. Chapter 3 uses tree decompositions as a way to automatically generate dynamic programming equations for RNA folding. Chapter 4 generalizes *covariance models*, a statistical model for families of structured homologs, thanks to treewidth-based algorithms for maximum inference.

1.2.4 Parameterized algorithmics in bioinformatics

Did you say NP-hard?. Parameterized algorithmics holds the promise that, when studying important problems, NP-hardness may not be the end of the story. It opens up the possibility of expanding

the scope of *exact solving* when instances have special structure.

Bioinformatics is a vast and ideal playground for parameterized algorithmics. Whether it is for RNA topics or other fields (such as phylogenetics [90, 91, 92]), many hard natural problems emerge from natural fundamental questions. Yet, actual biological data may exhibit simpler structures than the constructions used in NP-hardness reductions, which can be quantified in parameterizations. Historically, bioinformatics has played a major role in the development of parameterized algorithmics [93, 94]. A review of application areas of parameterized algorithmics within bioinformatics may be found in [95].

Parameterized RNA algorithmics. However, when it comes to applications to structural RNA bioinformatics, the only examples prior to this PhD are, to the author's knowledge, [58, 59, 62, 96, 48, 87, 97, 63]. In addition, note that some of these works, such as [63], do not explicitly state things in terms of parameterized algorithms, even though it is a way of describing the algorithms they develop. This surprising under-representation of RNA topics in applications of parameterized algorithmics, compared to other bioinformatics fields, is one of the main motivations for this PhD thesis.

Organization of this PhD thesis

Outline. Overall, the chapters of this thesis cover the following topics:

- Chapter 2 addresses the problem of removing a minimal number of edges in a graph to reach a target treewidth value. The resulting algorithm allows for the computation of *hierarchies* of simplifications of RNA structures. This could enable a hierarchical filtering approach for *pseudoknotted homolog search*, and a rejection-based acceleration of the algorithm of [48] for RNA DESIGN. It is based on [98], which is the journal version of [99].

- Chapter 3 explores the possibility of using tree decompositions as an *automated generation tool* for dynamic programming schemes tackling the RNA FOLDING problem. It provides a framework that, given the specification of a family of structures as a pseudoknotted conflict pattern (fatgraph), generates a dynamic programming scheme that solves RNA FOLDING restricted to this family. Interestingly, the computational complexity of the generated scheme is directly connected to the treewidth, and therefore minimized. It is based on the journal version of [100], which has been accepted but not published at the time of this writing.

- Chapter 4 focuses on methods and algorithms for *ncRNA homolog search*. It presents in particular a set of benchmark experiments for quantifying the added value of pseudoknot-aware alignments for STRUCTURE-SEQUENCE ALIGNMENT, such as [62]. It also gives possible formulations for *pseudoknotted covariance models*, based on tree decompositions. They can be seen both as extensions of [62] to include the features of `InfeRNAl`, the current pseudoknot-free state of the art solution, and as applications of the theory developed in Chapter 3. This is the only chapter that is not based on published materials.

- Chapter 5 is an account of a parameterized algorithmics study of the RNA ENERGY BARRIER problem. It introduces two parameters for the problem, namely the range of allowed energy values and the smallest number of hairpin loops in the two input structures. XP algorithms are formulated for both parameters. Interesting connections are drawn with a width parameter called *directed pathwidth*, as well as *independent-set reconfiguration* and *scheduling* problems. Many questions on this topic remain open, making these newly-drawn connections all the more interesting. This chapter is based on the journal version of [101], and has yet to be submitted.

Thesis by articles. Except for Chapter 4, all chapters in this thesis are based on published articles. A complete list of publications written as part of this PhD is given on the next page, along with their relationships to the chapters of this thesis.

List of publications

1. (Chapter 2) Bertrand Marchand, Yann Ponty, and Laurent Bulteau. Tree diet: reducing the treewidth to unlock FPT algorithms in RNA bioinformatics. *Algorithms for Molecular Biology*, 17(1):1–17, 2022

   ↪ **journal version of** Bertrand Marchand, Yann Ponty, and Laurent Bulteau. Tree Diet: Reducing the Treewidth to Unlock FPT Algorithms in RNA Bioinformatics. In *21st International Workshop on Algorithms in Bioinformatics*, volume 7, page 118, 2021

2. Laurent Bulteau, Bertrand Marchand, and Yann Ponty. A new parametrization for independent set reconfiguration and applications to RNA kinetics. In *IPEC 2021-International Symposium on Parameterized and Exact Computation*, 2021

   ↪ Chapter 5 is the tentative journal version of this paper [102], not submitted yet.

3. (Chapter 3) Bertrand Marchand, Sebastian Will, Sarah Berkemer, Laurent Bulteau, and Yann Ponty. Automated design of dynamic programming schemes for RNA folding with pseudoknots. *Algorithms for Molecular Biology*, 2023

   ↪ **journal version of** Bertrand Marchand, Sebastian Will, Sarah J Berkemer, Laurent Bulteau, and Yann Ponty. Automated Design of Dynamic Programming Schemes for RNA Folding with Pseudoknots. In *22nd International Workshop on Algorithms in Bioinformatics*, 2022

Chapter 2

TREE DIET: reducing the treewidth to unlock parameterized algorithms in RNA bioinformatics

This chapter is based on:

Bertrand Marchand, Yann Ponty, and Laurent Bulteau. Tree diet: reducing the treewidth to unlock FPT algorithms in RNA bioinformatics. *Algorithms for Molecular Biology*, 17(1):1–17, 2022

↪ **conference version** (WABI '21): https://drops.dagstuhl.de/opus/volltexte/2021/14360/pdf/LIPIcs-WABI-2021-7.pdf

↪ **journal version** (published): https://almob.biomedcentral.com/articles/10.1186/s13015-022-00213-z

Abstract

As touched upon in the introduction, treewidth-based algorithms have an extremely wide range of applicability, underlined by Courcelle's theorem. Unsurprisingly, many Bioinformatics problems lends themselves to this approach, including RNA DESIGN and STRUCTURE-SEQUENCE ALIGNMENT. Their time/space complexities depend critically on the treewidth of the input graphs, and these algorithms can typically only be run in practice up to a threshold treewidth value.

In order to extend their scope of applicability, this chapter introduces the TREE-DIET problem, i.e. the removal of a minimal set of edges such that a given tree-decomposition can be slimmed down to a prescribed treewidth tw'. The rationale is that the time gained thanks to a smaller treewidth in a parameterized algorithm compensates the extra post-processing needed to take deleted edges into account.

The core result is an FPT dynamic programming algorithm for Tree-Diet, using $2^{O(tw)}n$ time and space. It is complemented with parameterized complexity results regarding smaller parameters (e.g., NP-hardness when tw' or $tw - tw'$ is constant). A C++ prototype for our FPT algorithm in tw has been implemented. It allows to demonstrate potential applications to difficult instances of selected RNA-based problems: RNA design, Structure-Sequence Alignment, and search of pseudoknotted RNAs in genomes, revealing very encouraging results.

Contents

2.1 Introduction

Parameterized algorithmics in bioinformatics. Graph models and parameterized algorithms are found at the core of a sizable proportion of algorithmic methods in bioinformatics addressing a wide array of subfields, spanning sequence processing [104], structural bioinformatics [105], comparative genomics [106], phylogenetics [107], and further examples that can be found in a review by Bulteau and Weller [108].

Within RNA bioinformatics. RNA bioinformatics is no exception, with the prevalence of the secondary structure, an outer planar graph [109], as an abstraction of RNA conformations, and the notable utilization of graph models to represent complex topological motifs called pseudoknots [110], inducing the hardness of several tasks, such as structure prediction [111, 112, 113], structure alignment [114], or structure/sequence alignment [62]. Such motifs are functionally important and conserved, as witnessed by their presence in the consensus structure of 336 RNA families in the 14.5 edition of the RFAM database [115]. Moreover, methods in RNA bioinformatics [116] are increasingly

considering non-canonical base pairs and modules [117, 118], further increasing the density of RNA structural graphs and outlining the need for scalable algorithms.

Parameterized complexity: philosophy. As outlined in Chapter 1, a parameterized complexity approach can be used to circumvent the frequent NP-hardness of relevant problems. It generally considers one or several parameters, whose values are naturally bounded (or much smaller than the input size) within real-life instances. Once relevant parameters have been identified, one aims to design a Fixed Parameter Tractable (FPT) algorithm, having polynomial complexity for any fixed value of the parameter, and reasonable dependency on the parameter value.

Treewidth. The treewidth (section 1.2.3) is a classic parameter for FPT algorithms, and intuitively captures a notion of distance of the input to a tree. It is popular in bioinformatics due to the existence of efficient heuristics [119, 81] for computing tree-decompositions of reasonable treewidth. Given a tree-decomposition, many combinatorial optimization tasks can be solved using dynamic programming (DP), in time/space complexities that remain polynomial for any fixed treewidth value. Resulting algorithms remain correct upon (almost) arbitrary modifications of the objective function parameters, and can be adapted to study statistical properties of search spaces through changes of algebra.

FPT $\neq$practical. Unfortunately, the existence of a parameterized (or FPT) algorithm does not necessarily imply that of a practically-efficient implementation, even when the parameter takes low typical values. Indeed, the dependency of the complexity on the treewidth may be prohibitive, both in terms of time and memory requirements. This limitation is particularly obvious while searching and aligning structured RNAs, giving rise to an algorithmic problem called RNA structure-sequence alignment [120, 121, 62], for which the best known exact algorithm is in $\Theta(n.m^{tw+1})$, with n the structure length, m the sequence/window length, and tw the treewidth of the structure (inc. backbone). Such a complexity becomes impractical for structures having a treewidth higher than $\gtrsim 4$, which represent 50 to 60% of known RNA structures, as shown by Figure 1.11 in Chapter 1, based on a broad analysis of structures found in the PDB database. Similar complexities hold for problems that can be expressed as (weighted) constraint satisfaction problems, with m representing the cardinality of the variable domains. Such frameworks are frequently used for molecular design, both in proteins [122] and RNA [123], and may require the consideration of tree-widths as high as 20 or more [124].

Core idea of this Chapter. In this chapter, we investigate a pragmatic strategy to increase the practicality of parameterized algorithms based on the treewidth parameter [77]. We put our instance graphs on a diet, i.e. we introduce a preprocessing that reduces their treewidth to a prescribed value by removing a minimal cardinality set of edges. As discussed previously, the practical complexity of many algorithms greatly benefits from the consideration of simplified instances, having lower treewidth. Moreover, specific countermeasures for errors introduced by the simplification can sometimes be used to preserve the correctness of the algorithm. For instance, for searching structured RNAs using RNA structure-sequence alignment [120], an iterated filtering strategy could use instances of

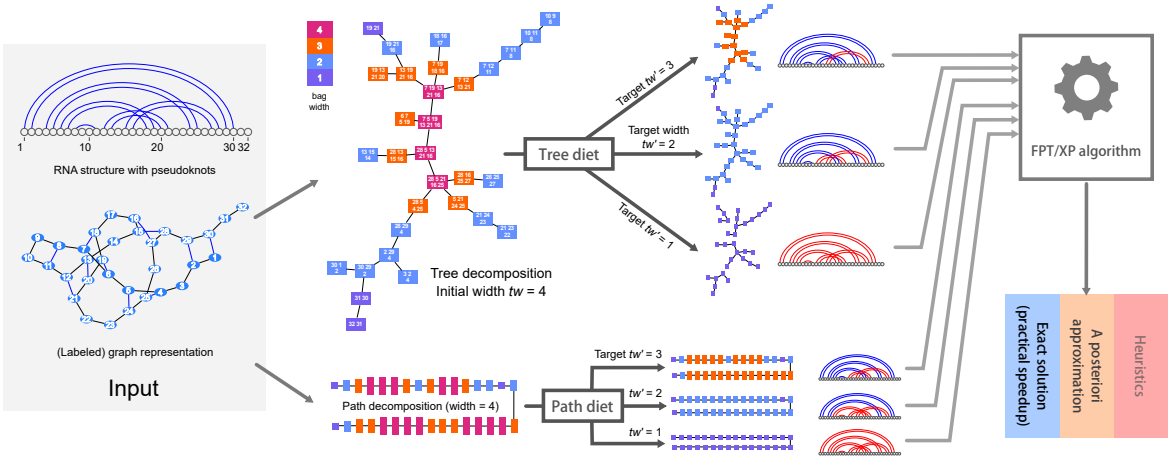


Figure 2.1: General description of our approach and rationale. Starting from a structured instance, e.g. an RNA structure with pseudoknots, our tree-diet/path-diet algorithms extract simplified tree/path decompositions, having prescribed target width tw'. Those can be used within existing parameterized algorithms to yield efficient heuristics, *a posteriori* approximations or even exact solutions.

increasing treewidth to restrict potential hits, weeding them early so that a – costly – full structure is reserved to (quasi-)hits. This strategy could remain exact while saving substantial time. Alternative countermeasures could be envisioned for other problems, such as a rejection approach to correct a bias introduced by simplified instances in RNA design. An overview of our approach is sketched on Figure 2.1

Chapter outline. After stating our problem(s) in Section 2.2, we study in Section 2.3 the parameterized complexity of the Graph-Diet problem, the removal of edges to reach a prescribed treewidth. We propose, in Section 2.4, a practical Dynamic Programming FPT algorithm for Tree-Diet, along with possible further optimizations for Path-Diet, two natural simplifications of the Graph-Diet problem, where a tree (resp. path) decomposition is provided as input and used as a guide. Finally, we show in Section 2.5 how our algorithm can be used to extract hierarchies of graphs/structural models of increasing complexity to provide alternative sampling strategies for RNA design, and speed-up the search for pseudoknotted non-coding RNAs. We conclude in Section 3.7 with future considerations and open problems.

2.2 Statement of the problem(s) and results

Reminder: tree decompositions. As introduced in section 1.2.3, a *tree-decomposition* $\mathcal{T}$ (over a set V of vertices) is a tree whose nodes are subsets of V, known as bags. The bags containing any $v \in V$ induce a (connected) subtree of $\mathcal{T}$. A *path-decomposition* is a tree-decomposition whose underlying tree $\mathcal{T}$ is a path. The *width* of $\mathcal{T}$ (denoted $w(\mathcal{T})$) is the size of its largest bag minus 1. An edge

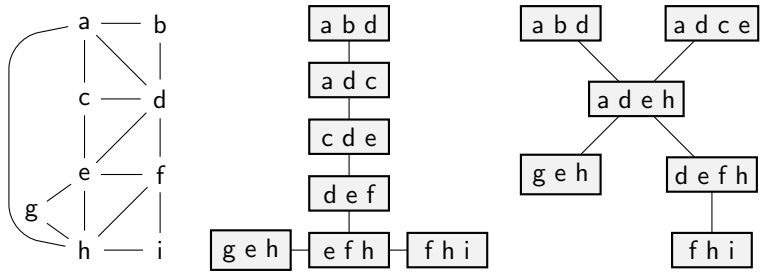


Figure 2.2: Illustrations for the Graph-Diet and Tree-Diet problems. Given a graph G on the left (treewidth 3), an optimal solution for Graph-Diet, with target treewidth 2, yields the tree-decomposition in the middle (edge ah is lost). On the other hand, any 1-tree-diet for the tree-decomposition on the right loses at least 3 edges.

$\{u, v\}$ is *visible* in $\mathcal{T}$ if some bag contains both u and v, otherwise it is *lost*. $\mathcal{T}$ is a *tree-decomposition of* G if all edges of G are visible in $\mathcal{T}$. The *treewidth* of a graph G is the minimum width over all tree-decompositions of G.

Problem 8 (Graph-Diet)**.** Given a graph $G = (V, E)$ of treewidth tw, and integers $tw' < tw$, find a tree-decomposition over V of width at most tw' losing a minimum number of edges from G.

A *tree-diet* of $\mathcal{T}$ is any tree-decomposition $\mathcal{T}'$ obtained by removing vertices from the bags of $\mathcal{T}$. $\mathcal{T}'$ is a d-tree-diet if $w(\mathcal{T}') \leq w(\mathcal{T}) - d$.

Problem 9 (Tree-Diet)**.** Given a graph G, a tree-decomposition $\mathcal{T}$ of G of width tw, and an integer $tw' < tw$, find a $(tw - tw')$-tree-diet of $\mathcal{T}$ losing a minimum number of edges.

Note that for Tree-Diet, $\mathcal{T}$ does not have to be optimal, so the width tw of the input tree decomposition might be larger than the actual treewidth of G, thus Tree-Diet can be used to reduce the width of *any* input decomposition. We define Binary-Tree-Diet and Path-Diet analogously, where $\mathcal{T}$ is restricted to be a binary tree (respectively, a path). An example of an instance of Graph-Diet and of Tree-Diet are given in Figure 2.2.

Reminder: parameterized complexity. The basics of parameterized complexity, introduced in section 1.2 can be loosely defined as follows (see [93] for the formal background). A *parameter* k for a problem is an integer associated with each instance which is expected to remain small in practical instances (especially when compared to the input size n). An exact algorithm, or the problem it solves, is FPT if it takes time $f(k)\text{poly}(n)$, and XP if it takes time $n^{g(k)}$ (for some functions f, g). Under commonly accepted conjectures (see for instance [71] for details), W[1]-hard problems may not be FPT, and Para-NP-hard problems (NP-hard even for some fixed value of k) are not FPT nor XP.

<table>
<tr><th>Parameter
Problem</th><th colspan="2">Source treewidth
tw</th><th>Target treewidth
tw'</th><th>Difference
$d = tw - tw'$</th></tr>
<tr><td>Graph-Diet</td><td colspan="2">FPT
via MSO
Theorem 5</td><td>Para-NP-hard
$tw' = 2$
EDP(K_4) [125]</td><td>Para-NP-hard*
$d = 1$
Theorem 6</td></tr>
<tr><td>Tree-Diet</td><td>XP
$O^*((6\Delta)^{tw})$
Theorem 11</td><td>FPT open</td><td rowspan="3">Para-NP-hard
$tw' = 1$
Theorem 7</td><td>Para-NP-hard
$d = 1$
Theorem 8</td></tr>
<tr><td>Binary-Tree-Diet</td><td colspan="2" rowspan="2">FPT
$O^*(12^{tw})$
Theorem 11</td><td rowspan="2">W[1]-hard
Theorem 9</td><td>XP open</td></tr>
<tr><td>Path-Diet</td><td>XP
$O^*(tw^d)$
Theorem 12</td></tr>
</table>

Table 2.1: Parameterized results for our problems. Algorithm complexities are given up to polynomial time factors (O^* notation), Δ denotes the maximum number of children in the input tree-decomposition, and "EDP" stands for Edge-Deletion Problem, as defined further down. (*) see Theorem 6 statement for a more precise formulation.

2.2.1 Our results

Our results are summarized in Table 2.1. Although the Graph-Diet problem would give the most interesting tree-decompositions in theory, it seems unlikely to admit efficient algorithms in practice (see Section 2.3).

Tree-diet. Thus we focus on the Tree-Diet relaxation, where an input tree-decomposition is given, which we use as a guide/restriction towards a thinner tree-decomposition. Seen as an additional constraint, it makes the problem harder (the case $tw' = 1$ becomes NP-hard, Theorem 7, although for Graph-Diet it corresponds to the Spanning Tree problem and is polynomial). With parameter tw however, it does help reduce the search space. In Theorem 11 we give an $O((6\Delta)^{tw}\Delta^2 n)$ Dynamic Programming algorithm, where Δ is the maximum number of children of any bag in the tree-decomposition. This algorithm can thus be seen as XP in general, but FPT on bounded-degree tree-decompositions (e.g., binary trees and paths). This is not a strong restriction, since the input tree may safely and efficiently be transformed into a binary one (see Supplementary Section A.1 for more details). Moreover, the duplications of bags which are used in the conversion may only decease the number of lost edges incurred by Tree-Diet.

We also consider the case where the treewidth needs to be reduced by $d = 1$ only, this without constraining the source treewidth. We give a polynomial-time algorithm for Path-Diet in this setting (Theorem 12) which generalizes into an XP algorithm for larger values of d, noting that an FPT algorithm for d is out of reach by Theorem 9. We also show that the problem is Para-NP-hard if the tree degree is unbounded (Theorem 8).

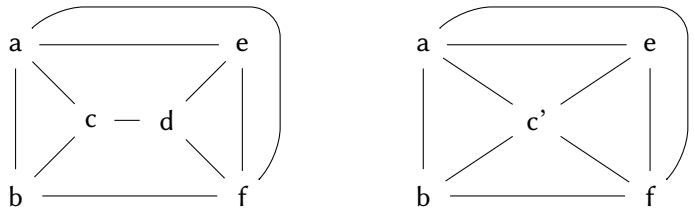


Figure 2.3: A graph G (left) with treewidth 3. Deleting edge cd gives treewidth 2, implying that $G \in$ Treewidth2 + 1e. However, if one contracts edge cd, then the resulting graph (right) has treewidth 3, and deleting any single edge does not decrease the treewidth. This example shows that the graph family Treewidth 2+1e is not minor-closed.

2.3 Algorithmic Limits: Parameterized Complexity Considerations

Graph-Diet as minimum edge deletion . Graph-Diet can be seen as a special case of the Edge Deletion Problem (EDP) for the family of graphs $\mathcal{H}$ of treewidth tw' or less: given a graph G, remove as few edges as possible to obtain a graph in $\mathcal{H}$. Such edge modification problems are more often parameterized by the number k of edited edges (see [126] for a complete survey). Given our focus on increasing the practicality of treewdith-based algorithms in bioinformatics, we restrict our focus to treewidth related parameters tw, tw' and $d = tw - tw'$.

Para-NP-hardness with tw'. Considering the target treewidth tw', we note that EDP is NP-hard when $\mathcal{H}$ is the family of treewidth-2 graphs [125], namely K_4-free graphs, hence the notation EDP(K_4). It follows that Graph-Diet is Para-NP-hard for the target treewidth parameter tw'.

2.3.1 Graph-Diet: practical solutions seem unlikely

Applying Courcelle's theorem and minor testing. For a combination of the parameters tw' and k, we could imagine graph minor theorems (such as Courcelle's theorem, Theorem 2) yielding parameterized algorithms "for free", as it is often the case with treewidth-based problems.

For tw'+k?. In this respect, Graph-Diet corresponds to deciding if a graph G belongs to the family of graphs having treewidth tw', augmented by k additional edges, denoted as Treewidth-tw'+ke since its introduction by Cai [127]. If this family were minor-closed, polynomial minor-free-testing [84, 86] would yield an FPT algorithm. However, this is not the case: for some graphs in the family, an edge contraction yields a graph G' not in Treewidth-tw'+ke, as illustrated by Figure 2.3.

For tw. Regarding the source graph treewidth tw, the vertex deletion equivalent of Graph-Diet, where one asks for a minimum subset of vertices to remove to obtain a given treewidth, is known as a Treewidth Modulator. This problem has been better-studied than its edge-deletion counterpart

[128], and has been shown to be FPT for the treewidth [129]. For the edge-deletion version (Graph-Diet), we can use an optimization variant of Courcelle's Theorem [71, Thm. 7.12] to show that the problem is FPT for tw. However, this is a purely theoretical result as the running-time of such "black-box" algorithms typically involve towers of exponentials on the treewidth parameter.

Theorem 5. Graph Diet *is* FPT *for the treewidth.*

Proof. We formulate Graph Diet as a Monadic Second-Order Logic (MSO) formula as follows: given a graph $G = (V, E)$, an integer tw' and a set X of edges, let $\phi_{tw'}(G, X)$ be true iff $G[E \setminus X]$ has treewidth tw'. Clearly $\phi_{tw'}$ can be expressed as an MSO formula, since both $G[E \setminus X]$ and "being of treewidth tw'" can be expressed in MSO [73]. Thus, by Arnborg et al. [130], there exists an algorithm that, given G of treewidth tw, finds a set X of minimum size satisfying $\phi_{tw'}(G, X)$ in time $f_{tw'}(tw) \cdot n$. Writing $g(tw) = \max_{tw' \leq tw} f_{tw'}(tw)$, this yields an algorithm for Graph Diet running in time at most $g(tw) \cdot n$. □

Impracticality of Graph-Diet. Overall, even though Graph Diet is FPT for the treewidth, "practical" exact algorithms seem out of reach. Indeed, any algorithm for Graph-Diet can be used to compute the Treewidth of an arbitrary graph, for which current state-of-the-art exact algorithms require time in $tw^{O(tw^3)}$ [77]. We thus have the following conjecture, which motivates the Tree-Diet relaxation of the problem.

Conjecture 1. Graph-Diet *does not admit algorithms with single-exponential running time for the treewidth.*

On a related note, it is worth noting that Edge Deletion to other graph classes (interval, permutation, ...) does admit efficient algorithms when parameterized by the treewidth alone [131], painting a contrasted picture.

Finally, for parameter d, any polynomial-time algorithm for constant d would allow to compute the treewidth of any graph in polynomial time. Since treewidth is NP-hard we have the following result.

Theorem 6. *There is no* XP *algorithm for* Graph-Diet *with parameter* d *unless* P= NP.

Proof. We consider the decision version of Graph-Diet where a bound k on the number of deleted edges is given. We build a Turing reduction from Treewidth: more precisely, assuming an oracle for Graph-Diet with $d = 1$ is available, we build a polynomial-time algorithm to compute the treewidth of a graph G. This is achieved by computing Graph-Diet$(G, tw, d = 1, k = 0)$ for decreasing values of tw (starting with $tw = |V|$): the first value of tw for which this call returns no solution is the treewidth of G. Note that this is not a *many-one* reduction, since several calls to Graph-Diet may be necessary (so this does not precisely qualify as an NP-hardness reduction, even though a polynomial-time algorithm for Graph-Diet$(G, tw, d = 1, k = 0)$ would imply P=NP). □

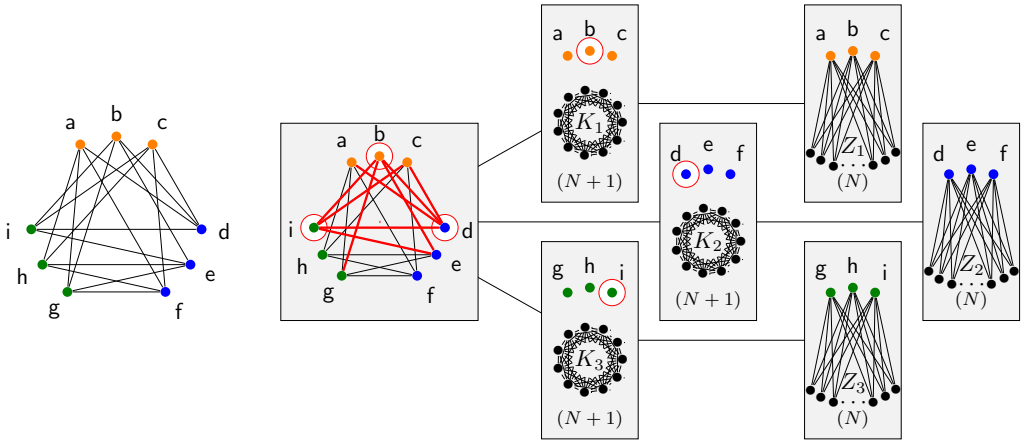


Figure 2.4: Reduction for Theorem 8 showing that Tree-Diet is NP-hard even for $d = 1$, from a graph G (left) with $k = 3$ and $n = 3$ to a graph G' (right, given by its tree-decomposition of width $N + n + 1$): a 1-tree-diet for G' amounts to selecting a k-clique in the root bag, i.e. in G.

2.3.2 Lower Bounds for Tree-Diet

Ruling out tw' and d. Parameters tw' and d would be the most interesting in practice, since parameterized algorithms would be efficient for small diets *or* small target treewidth. However, we prove strong lower-bounds for Tree-Diet on each of these parameters, leaving very little hope for parameterized algorithms (we thus narrow down the possible algorithms to the combined parameter $tw' + d$, i.e. tw, see Section 2.4). Only XP for parameter d when $\mathcal{T}$ has a constant degree remains open (cf. Table 2.1).

Theorem 7. Tree-Diet *and* Path-Diet *are* Para-NP*-hard for the target treewidth parameter* tw' *(*NP*-hard for* $tw' = 1$*).*

Proof. By reduction from the NP-hard problem Spanning Caterpillar Tree [132]: given a graph G, does G have a spanning tree C that is a caterpillar? Given $G = (V, E)$ with $n = |V|$, we build a tree-decomposition $\mathcal{T}$ of G consisting of $n-1$ bags containing all vertices (the width of the decomposition is therefore $n-1$) connected in a path. Then $(G, \mathcal{T})$ admits a tree-diet to treewidth 1 with $n-1$ visible edges if, and only if, G admits a caterpillar spanning tree. Indeed, the subgraph of G with visible edges must be a graph with pathwidth 1, i.e. a caterpillar [133]. With $n - 1$ visible edges, the caterpillar connects all n vertices together, i.e. it is a spanning tree. □

Theorem 8. Tree-Diet *is* Para-NP*-hard for parameter* d*. More precisely, it is* W[1]*-hard for parameter* Δ*, the degree of* $\mathcal{T}$*, even when* $d = 1$*.*

Proof. By reduction from Multi-Colored Clique (Given a graph G, an integer k and a partition of the vertices of G into k sets, is there a clique in G containing exactly one vertex from each of the k sets?). Consider a k-partite graph $G = (V, E)$ with $V = \bigcup_{i=1}^{k} V_i$. We assume that G is regular (each vertex has degree δ and that each V_i has the same size n (Multi Colored Clique is W[1]-hard under these restrictions [134, 135]). Let $L := \delta k - \binom{k}{2}$ and $N = \max\{|V|, L+1\}$. We now build a graph G' and a tree-decomposition $\mathcal{T}'$: start with $G' := G$. Add k independent cliques $K_1, \ldots, K_k$ of size $N+1$. Add k sets of N vertices Z_i ($i \in [k]$) and, for each $i \in [k]$, add edges between each $v \in V_i$ and each $z \in Z_i$. Build $\mathcal{T}$ using $2k+1$ bags $T_0, T_{1,i}, T_{2,i}$ for $i \in [k]$, such that $T_0 = V$, $T_{1,i} = V_i \cup K_i$ and $T_{2,i} = V_i \cup Z_i$. The tree-decomposition is completed by connecting $T_{2,i}$ to $T_{1,i}$ and $T_{1,i}$ to T_0 for each $i \in [k]$. Thus, $\mathcal{T}$ is a tree-decomposition of G' with $\Delta = k$ and maximum bag size $n + N + 1$ (vertices of V induce a size-3 path in $\mathcal{T}$, other vertices appear in a single bag, edges of G appear in T_0, edges of K_i in $T_{1,i}$, and finally edges between V_i and Z_i appear in $T_{2,i}$). The following claim completes the reduction:

$$\mathcal{T} \text{ has a 1-tree-diet losing at most } L \text{ edges from } G' \Leftrightarrow G \text{ has a } k\text{-clique.}$$

$\boxed{\Leftarrow}$ Assume G has a k-clique $X = \{x_1, \ldots, x_k\}$ (with $x_i \in V_i$). Build $\mathcal{T}'$ by removing each x_i from bags T_0 and $T_{1,i}$. Then $\mathcal{T}'$ is a 1-tree-diet of $\mathcal{T}$. There are no edges lost by removing x_i from $T_{1,i}$ (since x_i is not connected to K_i), and the edges lost in T_0 are all edges of G adjacent to any x_i. Since X forms a clique and each x_i has degree δ, there are $L = k\delta - \binom{k}{2}$ such edges.

$\boxed{\Rightarrow}$ Consider a 1-tree-diet $\mathcal{T}'$ of $\mathcal{T}$ losing L edges. Since each bag $T_{1,i}$ has maximum size, $\mathcal{T}'$ must remove at least one vertex x_i in each $T_{1,i}$. Note that $x_i \in V_i$ (since removing $x_i \in K_i$ would loose at least $N \geq L+1$ edges). Furthermore, x_i may not be removed from $T_{2,i}$ (otherwise N edges between x_i and Z_i would be lost), so x_i must also be removed from T_0. Let K be the number of edges in $G[\{x_1 \ldots x_k\}]$. The total number of lost edges in T_0 is $\delta k - K$. Thus, we have $\delta k - K \leq L$ and $K \geq \binom{k}{2}$: $\{x_1, \ldots, x_k\}$ form a k-clique of G. □

Theorem 9. Path-Diet *is* W[1]*-hard for parameter* d.

Proof. By reduction from Clique. Given a δ-regular graph G with n vertices and m edges and an integer k, consider the trivial tree-decomposition $\mathcal{T}$ of G with a single bag containing all vertices of G (it has width $n-1$). Then $(\mathcal{T}, G)$ has a k-tree-diet losing $\delta k - \binom{k}{2}$ edges if and only if G has a k-clique. Indeed, such a tree-diet $\mathcal{T}'$ would remove a set X of k vertices from G and losing $\delta k - \binom{k}{2}$ edges, so X induces $\binom{k}{2}$ edges and is a k-clique of G. Any instance G, with parameter k, of clique can therefore be transformed into an equivalent instance $(\mathcal{T}, G)$ of Path-diet, with parameter $d = k$. Since it qualifies as a parameterized reduction, Path-Diet is W[1]-hard. □

2.4 FPT Algorithm

2.4.1 For general tree-decompositions

We describe here a $O(3^{tw}n)$-space, $O(\Delta^{tw+2} \cdot 6^{tw}n)$-time dynamic programming algorithm for the Tree-Diet problem, with Δ and tw being respectively the maximum number of children of a bag in the input tree-decomposition and its width. On *binary* tree-decompositions (where each bag has at most 2 children), it yields a $O(3^{tw}n)$-space $O(12^{tw}n)$-time FPT algorithm.

Coloring formulation. We aim at solving the following problem: given a tree-decomposition $\mathcal{T}$ of width tw of a graph G, we want to remove vertices from the bags of $\mathcal{T}$ to reach a target width tw' while *losing* as few edges from G as possible. We tackle the problem through an equivalent *coloring* formulation: our algorithm will assign a color to each occurrence of a vertex in the tree decomposition. We work with three colors: red (r), orange (o), and green (g). Green means that the vertex is kept in the bag, while orange and red means removal of the vertex. An edge is thus visible within a bag when both its ends are green. It is lost if there is no bag where it is visible. To ensure equivalence with the original problem, the colors will be assigned following local rules, which we now describe.

Definition 18. A coloring of vertices in the bags of the decomposition is said to be *valid* if it follows the following rules:

- A vertex of a bag not present in its parent may be green or orange (R1)
- A green vertex in a bag may be either green or red in its children (R2)
- A red vertex in a bag must stay red in its children (R3)
- An orange vertex in a bag has to be either orange or green in exactly one child (unless there is no child with this vertex), and must be red in the other children (R4)

These rules are summarized in Figure 2.5 (a).

Rules rationale. When going down the tree, a green vertex may only stay green or permanently become red. As for orange vertices, they are locally absent but "may potentially be found further down the tree", while red vertices are removed from both the current bag and its entire subtree. An immediate consequence of these rules is therefore that the green occurrences of a given vertex form a (possibly empty) connected subtree. (R4) in particular is crucial to this connectivity: if an orange vertex could become orange in several children, it would be able to turn green in several disconnected subtrees. Figure 2.5 (b) shows an example sketch for a valid coloring of the occurrences of a given vertex in the tree-decomposition. A vertex may only be orange along a path starting form its highest occurrence in the tree, with any part branching off that path entirely red. It ends at the top of a (potentially empty) green subtree, whose vertices may also be parents to entirely red subtrees.

Formulations equivalence . We will now more formally prove the equivalence of the coloring formulation to the original problem. Let us first introduce two definitions. Given a valid coloring $\mathcal{C}$ of a tree-decomposition of G, an edge (u, v) of G is said to be *realizable* if there exists a bag in which both u and v are green per $\mathcal{C}$. Given an integer d, a coloring $\mathcal{C}$ of $\mathcal{T}$ is said to be $d-$*diet-valid* if removing red/orange vertices reduces the width of $\mathcal{T}$ from $w(T)$ to $w(T) - d$.

Proposition 2. *Given a graph G, a tree-decomposition $\mathcal{T}$ of width tw, and a target width $tw' < tw$, The* Tree-Diet *problem is equivalent to finding a $(tw - tw')$-diet-valid coloring $\mathcal{C}$ of $\mathcal{T}$ allowing for a number of realizable edges in G as large as possible.*

Proof. Given a $(tw - tw')$-tree-diet of $\mathcal{T}$ specifying which vertices are removed from which bags, we first show how to obtain a valid coloring $\mathcal{C}$ for $\mathcal{T}$ incurring the same number of lost (unrealizable) edges. Let us denote by $\mathcal{T}'$ the tree decomposition of width tw' obtained by applying the diet to $\mathcal{T}$. To start with, a vertex u is colored green in the bags where it is not removed. By the validity of $\mathcal{T}'$ as a decomposition, this set of bags forms a connected subtree, that we denote $\mathcal{T}_u^{\mathrm{g}}$. We also write $\mathcal{T}_u$ for the subtree of bags containing u in the original decomposition $\mathcal{T}$. If $\mathcal{T}_u^{\mathrm{g}}$ and $\mathcal{T}_u$ do not have the same root, then u is colored orange on the the path in $\mathcal{T}$ from the root of $\mathcal{T}_u$ (included) and the root of $\mathcal{T}_u^{\mathrm{g}}$ (excluded). Vertex u is colored red in any other bag of $\mathcal{T}_u$ not covered by these two cases. The resulting coloring follows rules (R1-4) and induces the same set of lost/non-realizable edges as the original $(tw - tw')$-tree-diet. Conversely, an equivalent $(tw - tw')$-tree-diet is obtained from a $(tw - tw')$-diet-valid coloring by removing red/orange vertices and keeping green ones. If a given vertex has no green occurrences, it is entirely removed from the tree decomposition and all its edges are *lost* (it becomes an isolated vertex). We may add it back to the tree decomposition by introducing a new bag containing only this vertex, which we connect arbitrarily to the tree decomposition. □

Decomposition of the search space and sub-problems. Based on this coloring formulation, we now describe a dynamic programming scheme for the Tree-Diet problem. We work with sub-problems indexed by tuples (X_i, f), with X_i a bag of the input tree decomposition and f a coloring of the vertices of X_i in green, orange or red (in particular, $f^{-1}(\mathtt{g})$ denotes the green vertices of X_i, and similarly for `o` and `r`).

Let us introduce some notations before giving the definition of our dynamic programming table. Given an edge (u, v) of G, realizable when coloring a tree-decomposition $\mathcal{T}$ of G with $\mathcal{C}$, we write $\mathcal{T}_{uv}^{\mathrm{g}}$ the subtree of $\mathcal{T}$ in which both u and v are green. We denote by $\mathcal{T}_i$ the subtree of the decomposition rooted at X_i, and $C(i, f)$ the d-diet-valid colorings of $\mathcal{T}_i$ agreeing with f on i, with $d = tw - tw'$. Our dynamic programming table is then defined as:

$$c(X_i, f) = \begin{cases} \max\limits_{\mathcal{C} \in C(i,f)} \left| \left\{ \begin{array}{r} \text{Edges } (u,v) \text{ of } G, \text{ realizable within } \mathcal{T}_i \text{ colored with } \mathcal{C} \\ \text{such that } \mathcal{T}_{uv}^{\mathrm{g}} \text{ is entirely contained strictly below } X_i \end{array} \right\} \right| \\ \qquad\qquad \text{if } f \text{ assigns green to at most } tw' + 1 \text{ vertices} \\ -\infty \qquad\qquad \text{otherwise} \end{cases}$$

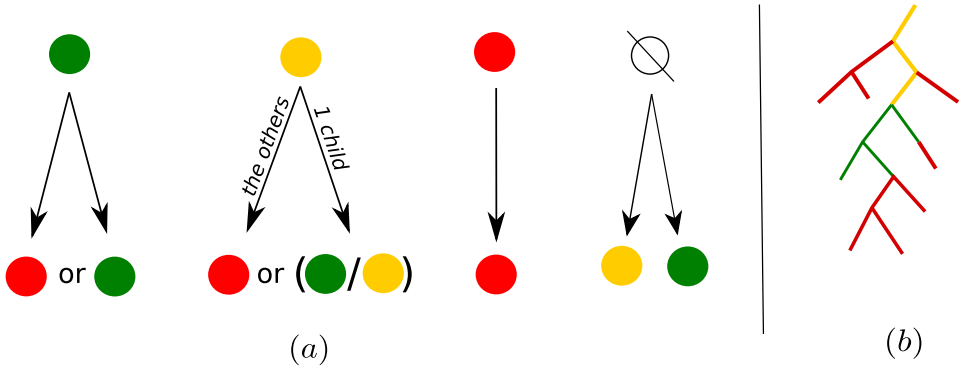


Figure 2.5: (*a*) Color assignation rules for vertices, when going down-tree. (*b*) Sketch of the general pattern our color assignation rules create on $\mathcal{T}_u$, the subtree of bags containing a given vertex u. Looking at it top-down: any orange part may only be a path starting at the root of the sub-tree. Some red sub-trees may branch off from it. On the sketch, at the end of the orange path, the vertex turns green. This top-most green vertex is at the root of a green sub-tree, with potential red sub-trees branching off from it.

The cell $c(X_i, f)$ therefore aggregates all edges realizable *strictly below* X_i. As we shall see through the recurrence relation below and its proof, edges with both ends green in X_i will be accounted for *above* X_i in $\mathcal{T}$.

We assume w.l.o.g that the tree-decomposition is rooted at an empty bag R. Given the definition of the table, the maximum number of realizable edges, compatible with a tree-diet of $(tw - tw')$ to $\mathcal{T}$, can be found in $c(R, \emptyset)$.

The following theorem presents a recurrence relation obeyed by $c(X_i, f)$:

Theorem 10. *For a bag X_i of $\mathcal{T}$, with children $Y_1, ...Y_\Delta$, we have:*

$$c(X_i, f) = \max_{m : f^{-1}(o) \to [1..\Delta]} \left[\sum_{1 \leq j \leq \Delta} \left(\max_{f'_j \in compatible(Y_j, f, m)} c(Y_j, f'_j) + \left| count(f, f'_j) \right| \right) \right]$$

with

- m*: a map from the orange vertices in X_i to the children of X_i. It decides for each orange vertex u, which child, among those which contain u, will color u orange or green; If there are no orange vertices in X_i, only the trivial empty map is considered.*
- $compatible(Y_j, f, m)$*: the set of colorings of Y_j compatible with f on X_i and m;*
- $count(f, f'_j)$*: set of edges of G involving two vertices of Y_j green by f'_j, but such that one of them is either not in X_i or not green by f.*

Note that $compatible(Y_j, f, m)$ may contain colorings f'_j that colour too many vertices in Y_j in green

to reach target width tw'. In that case $c(Y_j, f'_j) = -\infty$.

Theorem 10 relies on the following separation lemma for realizable edges under a valid coloring of a tree-decomposition. Recall that we suppose w.l.o.g that the tree-decomposition is rooted at an empty bag.

Lemma 1. *An edge (u, v) of G, realizable in $\mathcal{T}$ under $\mathcal{C}$, is contained in exactly one set of the form $count(C_{|P}, C_{|X})$ with X a bag of $\mathcal{T}$ and P its parent, $C_{|P}, C_{|X}$ the restrictions of $\mathcal{C}$ to P and X, respectively, and "count" defined as above. In addition, X is the root of the subtree of $\mathcal{T}$ in which both u and v are green.*

Proof. Given, in a tree-decomposition, a bag P colored with f, with a child X colored with h, a more precise definition for $count(f, h)$ is:

$$count(f,h) = \left\{ (u,v) \in E \;\middle|\; \begin{array}{c} h(u) = h(v) = \mathsf{g} \quad \text{and} \\ (u \notin P \text{ or } f(u) \neq \mathsf{g} \text{ or } v \notin P \text{ or } f(v) \neq \mathsf{g}) \end{array} \right\}$$

Now, given a realizable edge (u, v), in a tree-decomposition $\mathcal{T}$ colored with $\mathcal{C}$, the set of bags in which both u and v are green forms a connected subtree of $\mathcal{T}$. This subtree has a root, or *lowest common ancestor*, that we denote $R_{(u,v)}$. Since we assumed $\mathcal{T}$ to be rooted at an empty bag, $R_{(u,v)}$ is not the root of $\mathcal{T}$, and has a parent. We call this parent $P_{(u,v)}$. Clearly, (u, v) belongs to the "count set" associated to the edge $(P_{(u,v)}) \to (R_{(u,v)})$ of $\mathcal{T}$, while for any other edge $X \to Y$ of $\mathcal{T}$, the colors of u and v cannot verify the conditions to belong to the associated "count set". □

Proof of Theorem 10.

$\boxed{\leq}$ Let us more concisely use $RE_{\downarrow}(\mathcal{T}_i, \mathcal{C}, G)$ to denote the set of edges (u, v) of G, realizable under the $(tw - tw')$-diet-valid coloring $\mathcal{C}$ of $\mathcal{T}_i$, such that $\mathcal{T}^{\mathsf{g}}_{uv}$ is entirely contained strictly below X_i. We have, if f contains enough red/orange vertices to reduce the size of X_i to target size:

$$c(X_i, f) = \max_{\mathcal{C} \in C(i,f)} |RE_{\downarrow}(\mathcal{T}_i, \mathcal{C}, G)|$$

By definition, $c(X_i, f)$ is the maximum number of realizable edges in the subtree-decomposition rooted at X_i, such that all green-green occurrences of the edge occur strictly below X_i, and under the constraint that f colors X_i. Let $\mathcal{C}$ be a coloring for $\mathcal{T}_i$ realizing the optimum $c(X_i, f)$. Its restrictions to $Y_1 \dots Y_\Delta$ yield colorings $f'_1 \dots f'_\Delta$. Likewise, its restrictions to the subtree-decompositions $\mathcal{T}'_1 \dots \mathcal{T}'_\Delta$ rooted at $Y_1 \dots Y_\Delta$ yield colorings $\mathcal{C}'_1 \dots \mathcal{C}'_\Delta$ compatible with $f'_1 \dots f'_\Delta$. $\mathcal{C}'_1 \dots \mathcal{C}'_\Delta$ cannot be better than the optimal, so $\forall j, |RE_{\downarrow}(\mathcal{T}'_j, \mathcal{C}'_j, G)| \leq c(Y_j, f'_j)$

Let (u, v) be an edge of $RE_{\downarrow}(\mathcal{T}_i, \mathcal{C}, G)$. Per Lemma 6, either $(u, v) \in count(f, f'_j)$ for some j (if Y_j is the root of $\mathcal{T}^{\mathsf{g}}_{uv}$) and $(u, v) \notin \cup_j RE_{\downarrow}(\mathcal{T}'_j, \mathcal{C}'_j, G)$ or $(u, v) \in count(f, f'_j)$ and $\exists j$ such

that $(u, v) \in RE_{\downarrow}(\mathcal{T}'_j, \mathcal{C}'_j, G)$. Therefore:

$$\begin{aligned} c(X_i, f) = |RE_{\downarrow}(\mathcal{T}_i, \mathcal{C}, G)| &= \sum_{1 \leq j \leq \Delta} \left[|RE_{\downarrow}(\mathcal{T}'_j, \mathcal{C}'_j, G)| + count(f, f'_j) \right] \\ &\leq \sum_{1 \leq j \leq \Delta} \left(c(Y_j, f'_j) + count(f, f'_j) \right) \end{aligned}$$

and, a fortiori

$$c(X_i, f) \leq \max_{m : f^{-1}(\circ) \to [1 \ldots \Delta]} \sum_{1 \leq j \leq \Delta} \max_{f'_j \in compatible(Y_j, f, m)} \left(c(Y_j, f'_j) + count(f, f'_j) \right)$$

$\boxed{\geq}$ Conversely, given f, let m be an assignation map for orange vertices and $f'_1 \ldots f'_\Delta$ colorings of $Y_1 \ldots Y_\Delta$ compatible with f and m, and let $\mathcal{C}'_1 \ldots \mathcal{C}'_\Delta$ be colotings of $\mathcal{T}'_1 \ldots \mathcal{T}'_\Delta$ realizing the optima $c(Y_1, f'_1) \ldots c(Y_\Delta, f'_\Delta)$. The union of $\mathcal{C}'_1 \ldots \mathcal{C}'_\Delta$ and f is a coloring $\mathcal{C}$ for $\mathcal{T}_i$, the subtree-decomposition rooted at X_i, which can not be better than optimal ($|RE_{\downarrow}(\mathcal{T}_i, \mathcal{C}, G)| \leq c(X_i, f)$). As before, an edge (u, v) either belongs to $\cup_j count(f, f'_j)$ or to $\cup_j RE_{\downarrow}(\mathcal{T}'_j, \mathcal{C}'_j, G)$ but not both. In any case, it belongs to $RE_{\downarrow}(\mathcal{T}_i, \mathcal{C}, G)$. Therefore:

$$\begin{aligned} \sum_{1 \leq j \leq \Delta} \left(c(Y_j, f'_j) + count(f, f'_j) \right) &= \sum_{1 \leq j \leq \Delta} \left(|RE_{\downarrow}(\mathcal{T}'_j, \mathcal{C}'_j, G)| + count(f, f'_j) \right) \\ &= |RE_{\downarrow}(\mathcal{T}_i, \mathcal{C}, G)| \\ &\leq c(X_i, f) \end{aligned}$$

This is true for any choice of m, $f'_1 \ldots f'_\Delta$, therefore:

$$\max_{m : f^{-1}(\circ) \to [1 \ldots \Delta]} \sum_{1 \leq j \leq \Delta} \max_{f'_j \in compatible(Y_j, f, m)} \left(c(Y_j, f'_j) + count(f, f'_j) \right) \leq c(X_i, f)$$

which concludes the proof.

□

Dynamic programming algorithm. The recurrence relation of Theorem 10 naturally yields a dynamic programming algorithm for the Tree-Diet problem, as stated below:

Theorem 11. *There exists a $O(\Delta^{tw+2} \cdot 6^{tw} \cdot n)$-time, $O(3^{tw} \cdot n)$-space algorithm for the* Tree-Diet *problem, with Δ the maximum number of children of a bag in the input tree-decomposition, and tw its width.*

Proof of Theorem 11. Given the coloring formulation and Proposition 2, and given the sub-problems and $c(X_i, f)$-table definitions, with R the (empty) root of the tree-decomposition, $c(R, \emptyset)$ is indeed the maximum possible number of realizable edges when imposing a $(tw - tw')$-diet to $\mathcal{T}$. The recurrence relation of Theorem 10 therefore lends itself to a dynamic programming approach, over the tree-decomposition $\mathcal{T}$ following leaf-to-root order, for the problem.

It is reasonable to assume the number of bags in a tree decomposition to be linear in n (this is for instance the case for a *nice* tree decomposition [75, 71], or for a tree decomposition obtained from an *elimination ordering*, see [136, 119]). Therefore, the number of entries to the table is $O(3^{tw}n)$, given that a bag X may be colored in $3^{|X|}$ ways, and that the maximum size of X is $tw + 1$. For a given entry X_i, one must first enumerate all possible choices of $m : f^{-1}(\mathtt{o}) \to [1...\Delta]$, map assigning one child of X_i to each orange vertex in X_i. There are $O(\Delta^{tw+1})$ possibilities for m in the worst case, as $|f^{-1}(\mathtt{o})| \leq tw + 1$. Then, for each child Y_j, one must enumerate all possible colorings f'_j compatible with f. Possibilities for $f'_j(u)$ depend on the color by f:

- if $u \notin X_i \to f'_j(u) = \mathtt{o}$ or $\mathtt{g}$
- if $f(u) = \mathtt{g} \to f'_j(u) = \mathtt{g}$ or $\mathtt{r}$
- if $f(u) = \mathtt{o} \to f'_j(u) = \mathtt{o}$ or $\mathtt{g}$ if $m[u] = j$ or $f'_j(u) = \mathtt{r}$ otherwise.
- if $f(u) = \mathtt{r} \to f'_j(u) = \mathtt{r}$

Overall, as there are at most Δ children, $tw + 1$ vertices in each child, and 2 possibilities (see enumeration of cases above) of color for each vertex in a child, yielding a total number of compatible colorings bounded by $O(\Delta \cdot 2^{tw+1})$. Multiplying these contributions, the overall time complexity of our algorithm is therefore $O(\Delta^{tw+2} \cdot 6^{tw} \cdot n)$. □

Corollary 1. Binary-Tree-Diet *($\Delta = 2$) admits an* FPT *algorithm for the tw parameter.*

A pseudo-code implementation of the algorithm, using memoization, is included in Supplementary Section A.2

2.4.2 For path decompositions

Path decompositions. Let us start by defining path decompositions.

Definition 19. A path decomposition of a graph G is a tree decomposition $\mathcal{T} = (T, \{X_t\}_{t \in T}$ such that T is a *path*.

A path is a tree with only vertices of degree ≤ 2. Like treewidth, the *pathwidth* of a graph G is the smallest possible width of a path decomposition of G.

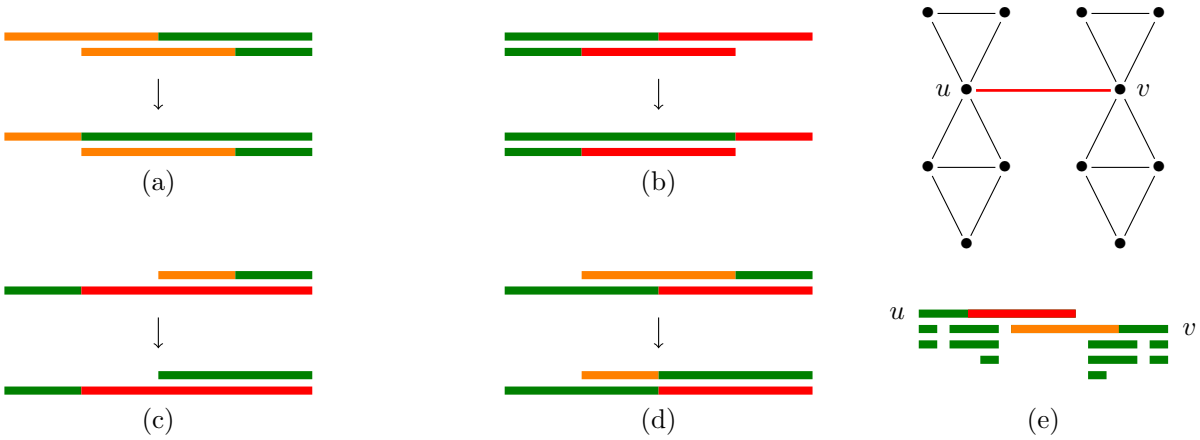


Figure 2.6: Five cases where two vertices are deleted in the same bag with $d = 1$. Bags are points in the line, and an interval covering all bags containing v is drawn for each v (with an equivalent coloring, see Proposition 2). Cases (a) to (d) can be safely avoided by applying the given transformations. In the example for case (e), however, it is necessary to delete both vertices u and v form a central bag. It is sufficient to avoid cases (a) and (b) in order to obtain an XP algorithm for d.

PATH-DIET. In this context of path decompositions, we note that the number of removed vertices per bag can be limited to at most $2d$ without losing the optimality. More precisely, we say that a coloring $\mathcal{C}$ is *d-simple* if any bag has at most d orange and d red vertices. We obtain the following result, using transformations given in Figure 2.6.

Proposition 3. *Given a graph G and a path-decomposition $\mathcal{T}$, if $\mathcal{C}$ is a d-diet-valid coloring of $\mathcal{T}$ losing k edges, then $\mathcal{T}$ has a d-diet-valid coloring that is d-simple, and loses at most k edges.*

Proof of Proposition 3. Consider such a coloring $\mathcal{C}$ with a maximal number of green vertices. We show that it is d-simple. Assume the path-decomposition $\mathcal{T}$ is rooted in bag X_1 and each X_i is the parent of X_{i+1}. Pick i to be the smallest index so that at least $d+1$ vertices in X_i are colored red by $\mathcal{C}$, assume any such i exists. Then one of these vertices, say u, is not colored red in X_{i-1} (either because $i = 1$, or it is not in X_{i-1}, or it is orange or green in X_{i-1}). Consider $\mathcal{C}'$ obtained by $\mathcal{C}$ and coloring u green in X_i. Then $\mathcal{C}'$ satisfies local rules R1 through R4 (a green vertex may be absent, green or orange in the parent bag, and a red vertex may be green in the parent bag). Furthermore, it is d-diet-valid since it still removes at least d (red) vertices in X_i. Overall $\mathcal{C}'$ is another d-diet-valid coloring with more green vertices: a contradiction, so no such i exist (and no bag has $d+1$ red vertices). The same argument works symmetrically for orange vertices. Overall, $\mathcal{C}$ is d-simple. □

Together with Proposition 2, this shows that it is sufficient to restrict our algorithm to d-simple colorings. (See also Figure 2.6). In particular, for any set X_i, choosing which $\leq d$ vertices are orange and which $\leq d$ are red, among the total of n vertices, is enough to fix a coloring. The number of

such colorings is therefore bounded by $O(tw^{2d})$. Applying this remark to our algorithm presented in Section 2.4.1 yields the following result:

Theorem 12. PATH-DIET *can be solved in* $O(tw^{2d}n)$*-space and* $O(tw^{4d}n)$*-time.*

2.5 Proofs of concept

Implementation. We now illustrate the relevance of our approach, and the practicality of our algorithm for TREE-DIET, by using it in conjunction with FPT algorithms for three problems in RNA bioinformatics. We implemented in C++ the dynamic programming scheme described in Theorem 11 and Supplementary Section A.2. Its main primitives are made available for Python scripting through pybind11 [137]. It actually allows to solve a generalized *weighted* version of TREE DIET, as explained in Supplementary Section A.2. This feature allows to favour the conservation of important edges (e.g., RNA backbone) during simplification, by assigning them a much larger weight compared to other edges. Our implementation is freely available at https://gitlab.inria.fr/amibio/tree-diet.

Execution time. The execution time of this implementation on elements of the data set used for Figure 1.11 (all RNA-only structures of the PDB database) is represented on Figure 2.7, for input treewidth values of up to 7. It shows that our tree-diet method is applicable with reasonable run-times ($\lesssim 1$ hour) for all structures of width ≤ 7. The proofs-of-concepts presented in this section involve however instances with treewidth of up to 9, in the case of RNA design, for which the run-time also stays reasonable.

2.5.1 Memory-parsimonious unbiased sampling of RNA designs

speeding up RNAPond. As a first use case for our simplification algorithm, we strive to ease the sampling phase of a recent method, called RNAPond [123], addressing RNA negative design. The method targets a set of base pairs S, representing a secondary structure of length n, and infers a set $\mathcal{D}$ of m disruptive base pairs (DBPs) that must be avoided. It relies on a $\Theta(k \cdot (n+m))$ time algorithm for sampling k random sequences (see Supplementary Section A.3 for details) after a preprocessing in $\Theta(n \cdot m \cdot 4^{tw})$ time and $\Theta(n \cdot 4^{tw})$ space. Here, the input consists of a graph $G = ([1, n], S \cup \mathcal{D})$ and a tree decomposition $\mathcal{T}$ of G, having width tw. In practice, the preprocessing largely dominates the overall runtime, even for large values of k, and its large memory consumption represents the main bottleneck.

Rejection sampling. This discrepancy in the complexities/runtimes of the preprocessing and sampling suggests an alternative strategy: relaxing the set of constraints to $(S', \mathcal{D}')$, with $(S' \cup \mathcal{D}') \subset (S \cup \mathcal{D})$, and compensating it through a rejection of sequences violating constraints in $(S, \mathcal{D}) \setminus (S', \mathcal{D}')$. The relaxed algorithm would remain unbiased, while the average-case time complexity of the rejection algorithm would be in $\Theta(k \cdot \overline{q} \cdot (n+m))$ time, where $\overline{q}$ represents the relative increase of the

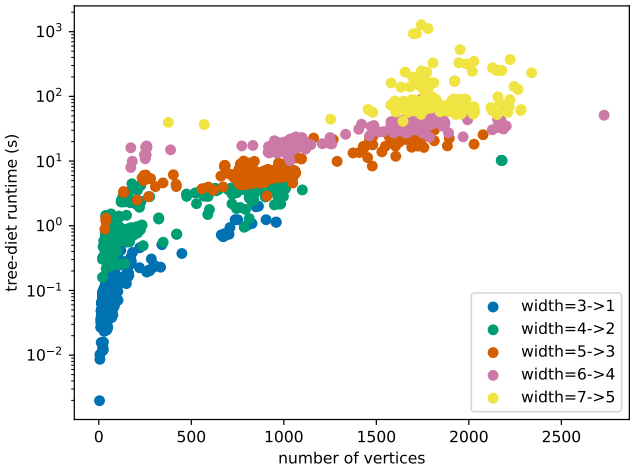


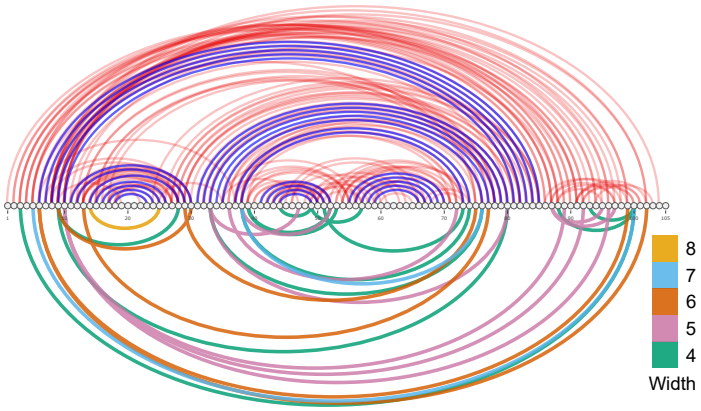
Figure 2.7: Run-time of the tree-diet algorithm on all RNA-only structures of the PDB database, versus the size (length of the RNA string) of these structures. The data set consists of RNA structures from the PDB [138] with structure graphs extracted with DSSR [22], limited to structures of treewidth ≤ 7. Structures are colored by their original treewidth. Here, we have asked the algorithm to reduce the treewidth by 2.

partition function ($\approx$ the sequence space) induced by the relaxation. The preprocessing step would retain the same complexity, but based on a (reduced) treewidth $tw' \leq tw$ for the relaxed graph $G' = ([1, n], S' \cup \mathcal{D}')$.

These complexities enable a trade-off between the rejection (time), and the preprocessing (space), which may be critical to unlock future applications of RNA design. Indeed, the treewidth can be decreased by removing relatively few base pairs, as demonstrated below using our algorithm on pairs inferred for hard design instances.

Practical evaluation. We considered sets of DBPs inferred by RNAPond over two puzzles in the EteRNA benchmark. The EteRNA22 puzzle is an empty secondary structure spanning 400 nts, for which RNAPond obtains a valid design after inferring 465 DBPs. A tree decomposition of the graph formed by these 465 DPBs is then obtained with the standard min-fill-ordering heuristic [81], giving a width of 6. The EteRNA77 puzzle is 105 nts long, and consists in a collection of helices interspersed with destabilizing internal loops. RNApond failed to produce a solution, and its final set of DBPs consists of 183 pairs, for which the same heuristic yields a tree decomposition of width 9. We further make both tree decompositions binary through bag duplications (see Supplementary Section A.1), giving an FPT runtime to our algorithm, while potentially lowering the number of lost edges.

Executing the tree-diet algorithm (Theorem 11) on both graphs and their tree decompositions, we obtained simplified graphs, having lower treewidth while typically losing few edges, as illustrated and reported in Figure 2.8. Remarkably, the treewidth of the DBPs inferred for EteRNA22 can be decreased



	#Preserved BPs	
tw'	EteRNA22	EteRNA77
9	–	**183**
8	–	182
7	–	180
6	**465**	176
5	460	168
4	456	157
3	445	144
2	418	121
1	320	86

Figure 2.8: (Left) Target secondary structure (blue BPs), full set of disruptive base pairs (DPB; top) inferred by RNAPond on the Eterna77 puzzle, and subsets of DBPs (bottom) cumulatively removed by the tree-diet algorithm to reach prescribed treewidths. (Right) Number of BPs retained by our algorithm, targeting various treewidth values for the EteRNA22 and EteRNA77 puzzles.

to $tw' = 5$ by only removing 5 DBPs/edges (460/465 retained), and to $tw' = 4$ by removing 4 further DBPs (456/465). For EteRNA77, our algorithm reduces the treewidth from 9 to 6 by only removing 7 DBPs.

Speed-up estimate. Rough estimates can be provided for the trade-off between the rejection and preprocessing complexities, by assuming that removing a DBP homogeneously increases the value of the partition function $\mathcal{Z}$ by a factor $\alpha := 16/10$ (#pairs/#incomp. pairs). The relative increase in partition function is then $\overline{q} \approx \alpha^b$, when b base pairs are removed. For EteRNA22, reducing the treewidth by 2 units (6→4), i.e. a 16 fold reduction of the memory and preprocessing time, can be achieved by removing 9 DBPs, i.e. a 69 fold expected increase in the time of the generation phase. For EteRNA77, the same 16 fold ($tw' = 9 \to 7$) reduction of the preprocessing time/space can be achieved through an estimated 4 fold increase of the generation time. A more aggressive 256 fold memory gain can be achieved at the expense of an estimated 1 152 fold increase in generation time. Given the large typical asymmetry in runtimes and implementation constants between the computation-heavy preprocessing and, relatively light, generation phases, the availability of an algorithm for the TREE-DIET problem provides new options, especially to circumvent memory limitations.

2.5.2 Structural alignment of complex RNAs

Structural homology is often posited within functional families of non-coding RNAs, and is foundational to algorithmic methods for multiple RNA alignments [115], considering RNA base pairs while aligning distant homologs. In the presence of complex structural features (pseudoknots, base triplets), the sequence-structure alignment problem becomes hard, yet admits XP solutions based on the treewidth of the base pair + backbone graph. In particular, Rinaudo *et al.* [62] describe a $\Theta(n.m^{tw+1})$

algorithm for optimally aligning a structured RNA of length n onto a genomic region of length m. It optimizes an alignment score that includes: i) substitution costs for matches/mismatches of individual nucleotides and base pairs (including arc-breaking) based on the RIBOSUM matrices [139]; and ii) an affine gap cost model [140]. We used the implementation of the Rinaudo *et al.* algorithm, implemented in the LicoRNA software package [141, 142].

Impact of treewidth on the structural alignment of a riboswitch. In this case study, we used our tree-diet algorithm to modulate the treewidth of complex RNA structures, and investigate the effect of the simplification on the quality and runtimes of structure-sequence alignments. We considered the Cyclic di-GMP-II riboswitch, a regulatory motif found in bacteria that is involved in signal transduction, and undergoes conformational change upon binding the second messenger c-di-GMP-II [143, 144]. A 2.5Å resolution 3D model of the c-di-GMP-II riboswitch in *C. acetobutylicum*, proposed by Smith *et al.* [145] based on X-ray crystallography, was retrieved from the PDB [88] (PDBID: 3Q3Z). We annotated its base pairs geometrically using the DSSR method [146]. The canonical base pairs, supplemented with the backbone connections, were then accumulated in a graph, for which we heuristically computed an initial tree decomposition $\mathcal{T}_4$, having treewidth $tw = 4$.

We simplified our the initial tree decomposition $\mathcal{T}_4$, and obtained simplified models $\mathcal{T}_3$, and $\mathcal{T}_2$, having width $tw' = 3$ and 2 respectively. As controls, we included tree decompositions based on the secondary structure (max. non-crossing set of BPs; $\mathcal{T}_{2D}$) and sequence ($\mathcal{T}_{1D}$). We used LicoRNA to predict an alignment $a_{\mathcal{T},w}$ of each original/simplified tree decomposition $\mathcal{T}$ onto each sequence w of the c-di-GMP-II riboswitch family in the RFAM database [115] (RF01786). Finally, we reported the LicoRNA runtime, and computed the Sum of Pairs Score (SPS) [147] as a measure of the accuracy of $a_{\mathcal{T},w}$ against a reference alignment $a^\star_w$:

$$\mathsf{SPS}(a_{\mathcal{T},w}; a^\star_w) = \frac{|\,\mathsf{MatchedCols}(a_{\mathcal{T},w}) \cap \mathsf{MatchedCols}(a^\star_w)\,|}{|\,\mathsf{MatchedCols}(a^\star_w)\,|},$$

using as reference the alignment $a^\star_w$ between the 3Q3Z sequence and w induced by the manually-curated RFAM alignment of the RF01786 family.

Results. The results, presented in Figure 2.9, show a limited impact of the simplification on the quality of the predicted alignment, as measured by the SPS in comparison with the RFAM alignment. The best average SPS (77.3%) is achieved by the initial model, having treewidth of 4, but the average difference with simplified models appears very limited (e.g., 76.5% for $\mathcal{T}_3$), especially when considering the median. Meanwhile, the runtimes mainly depend on the treewidth, ranging from 1h for $\mathcal{T}_4$ to 300ms for $\mathcal{T}_{1D}$. Overall, $\mathcal{T}_{2D}$ seems to represent the best compromise between runtime and SPS, although its SPS may be artificially inflated by our election of RF01786 as our reference (built from a covariance model, i.e. essentially a 2D structure). Finally, the difference in number of edges (and induced SPS) between $\mathcal{T}_{2D}$ and $\mathcal{T}_2$, both having $tw = 2$, exemplifies the difference between the Tree-Diet and Graph-Diet problems, and motivates further work on the latter.

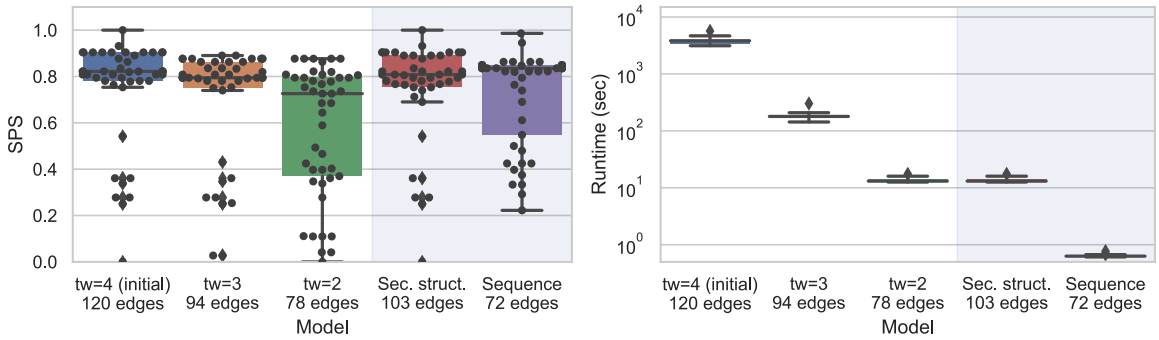


Figure 2.9: Impact on alignment quality (SPS; Left) and runtime (Right) of simplified instances for the RNA sequence-structure alignment of the pseudoknotted c-di-GMP-II riboswitch. The impact of simplifications on the quality of predicted alignments, using RFAM RF01786 as a reference, appears limited while the runtime improvement is substantial.

Exact iterative strategy for the genomic search of ncRNAs. In this final case study, we consider an exact filtering strategy to search new occurrences of a structured RNA within a given genomic context. In this setting, one attempts to find all ε-admissible (cost $\leq \varepsilon$) occurrences/hits of a structured RNA S of length n within a given genome of length $g \gg n$, broken down in windows of length $\kappa.n$, $\kappa > 1$. Classically, one would align S against individual windows, and report those associated with an $\epsilon-$admissible alignment cost. This strategy would have an overall $\Theta(g \cdot n^{tw+2})$ time complexity, applying for instance the algorithm of [62].

Our instance simplification framework enables an alternative strategy, that incrementally filters out unsuitable windows based on models of increasing granularity. Indeed, for any given target sequence, the min alignment cost c_δ obtained for a simplified instance of treewidth $tw - \delta$ can be corrected (*cf* Supplementary Section A.4) into a lower bound $c^\star_\delta$ for the min alignment cost $c^\star_0$ of the full-treewidth instance tw. Any window such that $c^\star_\delta > \varepsilon$ thus also obeys $c^\star_0 > \varepsilon$, and can be safely discarded from the list of putative ε-admissible windows, without having to perform a full-treewidth alignment. Given the exponential growth of the alignment runtime for increasing treewidth values (see Figure 2.9-right) this strategy is expected to yield substantial runtime savings.

Application: twister ribozyme. We used this strategy to search occurrences of the Twister ribozyme (PDBID 4OJI), a highly-structured ($tw = 5$) 54nts RNA initially found in *O. sativa* (Asian rice) [10]. We targeted the *S. bicolor* genome (sorghum), focusing on a 10kb region centered on the 2,485,140 position of the 5th chromosome, where an instance of the ribozyme was suspected within an uncharacterized transcript (LOC110435504). The 4OJI sequence and structure were extracted from the 3D model as above, and included into a tree decomposition $\mathcal{T}_5$ (73 edges), simplified into $\mathcal{T}_4$ (71 edges), $\mathcal{T}_3$ (68 edges) and $\mathcal{T}_2$ (61 edges) using the tree-diet algorithm.

We aligned all tree decompositions against all windows of size 58nts using a 13nts offset, and measured the score and runtime of the iterative filtering strategy using a cost cutoff $\varepsilon = -5$. The

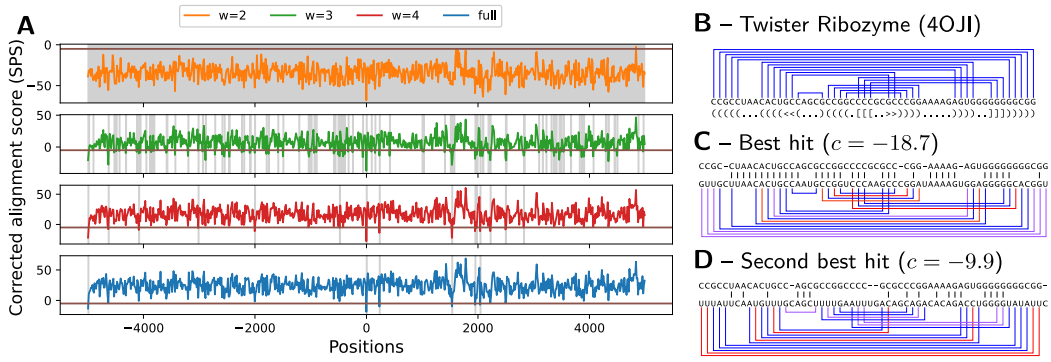


Figure 2.10: Corrected costs associated with the search for structured homologs of the Twister ribozyme in chromosome 5 of *S. bicolor*, using simplified instances of various treewidth (A). Gray areas represent scores which, upon correction, remain below the cutoff, and have to be considered for further steps of the iterated filtering. Canonical base pairs of the ribozyme (PDBID 4OJI; B), mapped onto to the best hit (C) and second best hit (D) found along the search colored depending on their support in the target sequence (Red: incompatible; Purple: unstable G-U; Blue: stable).

search recovers the suspected occurrence of twister as its best result (Figure 2.10.C), but produced hits (*cf* Figure 2.10.D) with comparable sequence conservation that could be the object of further studies. Regarding the filtering strategy, while $\mathcal{T}_2$ only allows to rule out 3 windows out of 769, $\mathcal{T}_3$ allows to eliminate an important proportion of putative targets, retaining only 109 windows, further reduced to 15 windows by $\mathcal{T}_4$, 6 of which end up as final hits for the full model $\mathcal{T}_5$ (*cf* Figure 2.10.A). The search remains exact, but greatly reduces the overall runtime from 24 hours to 34 minutes (42 fold!).

2.6 Conclusion and discussion

We have established the parameterized complexity of three treewidth reduction problems, motivated by applications in Bioinformatics, as well as proposed practical algorithms for instances of reasonable treewidths. The reduced widths obtained by our proposed algorithm can be used to obtain: i) sensitive heuristics, owing to the consideration of a maximal amount of edges/information in the thinned graphs; ii) *a posteriori* approximation ratios, by comparing the potential contribution of removed edges to the optimal score obtained of the thinned instance by a downstream FPT/XP algorithm; iii) substantial practical speedups without loss of correctness, e.g. when partial filtering can be safely achieved based on simplified input graphs.

2.6.1 Open questions

Regarding the parameterized complexity of Graph-Diet and Tree-Diet, some questions remain open (see Table 2.1): an FPT algorithm for Tree-Diet (ideally, with $2^{O(tw)} \cdot n$ running time), would be the most desirable, if possible satisfying the backbone constraints. The existence of such an algorithm is

not trivial. In particular, it is perhaps worth noting that it is not implied by the existence of an FPT algorithm for GRAPH-DIET with the input treewidth as a parameter (5). Indeed, in comparison to the latter, TREE-DIET subtly restricts the search space to tree decompositions that are subsets of the input tree decomposition. It follows that the result of GRAPH DIET for a graph G may substantially differ from the result of TREE-DIET given a tree decomposition $\mathcal{T}$ of G as input. We also aim at trying to give efficient exact algorithms for GRAPH DIET in the context of RNA (we conjecture this is impossible in the general case). Finally, we did not include the number of deleted edges in our multivariate analysis: even though in practice it is more difficult *a priori* to guarantee their small number, we expect it can be used to improve the running time in many cases.

2.6.2 Backbone Preservation.

In two of our applications, the RNA secondary structure graph contains two types of edges: those representing the *backbone* of the sequence (i.e., between consecutive bases) and those representing base pair bonds. In practice, we want all backbone edges to be visible in the resulting tree-decomposition, and only base pairs may be lost. This can be integrated to the TREE-DIET model (and to our algorithms) using weighted edges, using the total weight rather than the count of deleted edges for the objective function. Note that some instances might be unrealizable (with no tree-diet preserving the backbone, especially for low tw'). In most cases, ad-hoc bag duplications can help avoid this issue. The design of pre-processing methods, involving bag duplications or other operations on tree decompositions, and aimed at ensuring the existence of a backbone-preserving tree-diet will be the subject of future work.

Chapter 3

Automated design of dynamic programming schemes for RNA folding with pseudoknots

This chapter is based on:

Bertrand Marchand, Sebastian Will, Sarah Berkemer, Laurent Bulteau, and Yann Ponty. Automated design of dynamic programming schemes for RNA folding with pseudoknots. *Algorithms for Molecular Biology*, 2023

↪ **conference version (WABI '22)**: https://drops.dagstuhl.de/opus/volltexte/2022/17041/pdf/LIPIcs-WABI-2022-7.pdf

↪ **journal version (accepted)**: https://hal.science/hal-04103565

Abstract

Although RNA secondary structure prediction is a textbook application of dynamic programming (DP) and routine task in RNA structure analysis, it remains challenging whenever pseudoknots come into play. Since the prediction of pseudoknotted structures by minimizing (realistically modelled) energy is NP-hard, specialized algorithms have been proposed for restricted conformation classes that capture the most frequently observed configurations. To achieve good performance, these methods rely on specific and carefully hand-crafted DP schemes.

In contrast, we generalize and fully automatize the design of DP pseudoknot prediction algorithms. For this purpose, 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 (and even possibly $\mathcal{O}(n^{tw})$ in simple energy models) for predicting the MFE folding of an RNA of length n. We demonstrate, for the most common pseudoknot classes, that our automatically generated algorithms achieve the same complexities as reported in the literature for hand-crafted schemes.

Our 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.

Contents

3.1 Introduction

RNA folding. The function of non-coding RNAs is, to a large extent, determined by their structure. Structure prediction algorithms therefore play a crucial role in biomedical 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` [24], `RNAfold` [25], `RNAstructure` [26]. A recent alternative to predictions based on experimentally determined energy

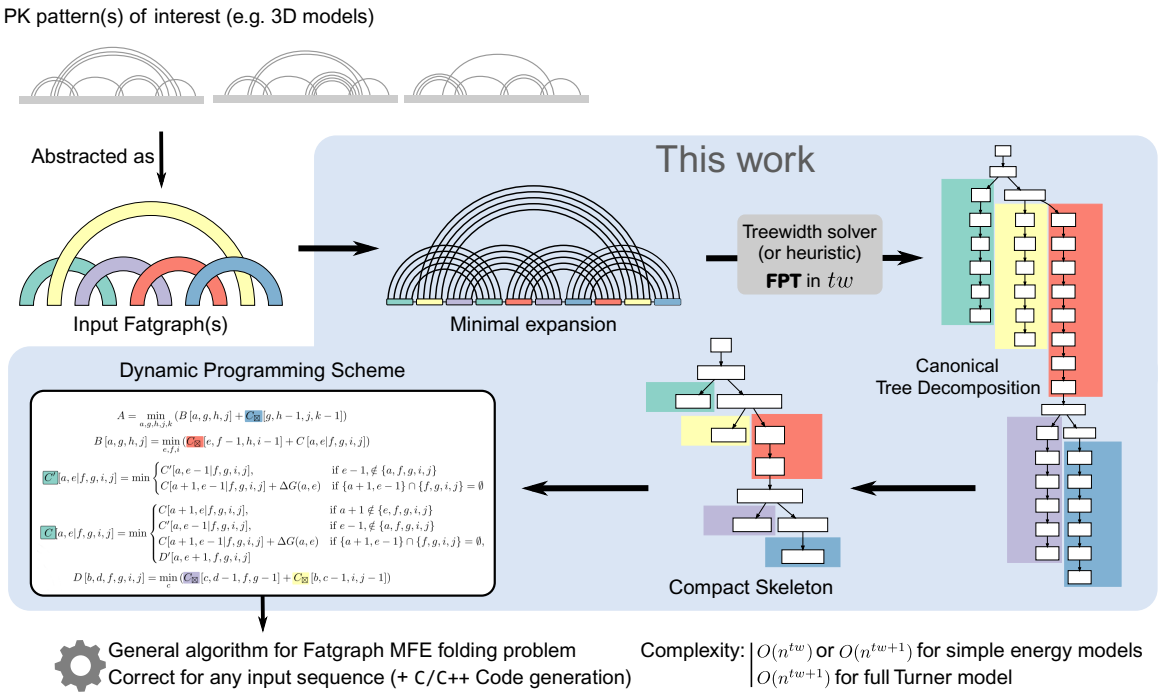


Figure 3.1: Given a finite number of arbitrary fatgraphs, a dynamic programming scheme for folding (restricted to the family of structures specified by the fatgraphs) is derived from canonical tree decompositions of minimal representative expansions of the helices, for each fatgraph. The workflow gives an overview of the steps of the algorithm. Each step is described in more details in the subsequent sections and figures: see Figure 3.2 for fatgraphs, Figure 3.5 and Section 3.3 for a detailed version of the canonical tree decomposition, Figure 3.8 for a detailed view of the compact skeleton of the tree decomposition.

parameters are machine learning approaches that train models on known secondary structures, e.g. `CONTRAfold` [148], `ContextFold` [149], `MXfold2` [150].

Pseudoknotted RNA folding. However, the most frequently used algorithms (including all of the above ones) optimize solely over pseudoknot-free structures [151], which do not contain crossing base pairs. Although pseudoknots (PK) appear in many RNA secondary structures, they have been omitted by initial prediction algorithms due to their computational complexity [152], and the difficulty to score individual conformations [153]. 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` [30], `NUPACK` [29], `gfold` [16], `Knotty` [45] or they use heuristics that don't guarantee exact solutions, e.g. `HotKnots` [154], `IPknot` [155, 150], `Hfold` [156].

Designing dynamic programming algorithms. Owing 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 [157], topologically bounded [16]) that need to be captured. A detailed overview of pseudoknot categories is given in [158]. Similar situations occur for RNA-RNA interactions [27], possibly including several RNA molecules. Interestingly, when more than two RNA strands are considered, existing algorithms restrict the joint conformation to crossing-free interactions [159], further motivating an, ideally-automated, design of algorithms beyond the case of pseudoknot-free secondary structures.

Tree decompositions as automation tool. The paradigm of tree decompositions (TD) represents an appealing candidate automating such a design task. TDs organize the vertices of a graph into a tree-like structure that represent all vertices and edges, augmented with a notion of consistency. A TD can then be re-interpreted as DP schemes for a wealth of graph problems involving local constraints (coloring, independent sets, covers...) [75] and complex pattern matching problems in Bioinformatics [62]. The complexities of such exact algorithms are typically exponential on a parameter called the treewidth, which can be minimized to obtain an optimal TD in time only exponential on the min treewidth itself [77]. However, TD-based approaches typically start from a single input graph, whereas folding prediction requires DP schemes that generalize to collections of structures of unbounded cardinalities. This led us to the following question, at the foundation of this work:

> *Can tree decompositions be used to infer structure prediction algorithms that work for entire classes of conformations?*

Results. In this work, we answer positively to that question. We consider popular classes of pseudoknotted structures, described as fatgraphs [160, 16, 161, 162], an abstraction of RNA conformations related to RNA shapes [163] or shadows [157, 16]. 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 corresponding to 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 methodology leads to a generalization of algorithms underlying `LiCoRNA` [164] and `gfold` [16] and represents a parameterized algorithm based on the treewidth (tw) of the underlying fatgraph.

`gfold` generalization. For example, our method automatically derives optimally efficient recursions of a `gfold`-like prediction algorithm covering the four pseudoknot types of 1-structures (cf

Table 3.2) Moreover, it enables highly complex implementations, like a prediction algorithm for 2-structures. Notably, this was never implemented for `gfold`, since it requires the generation of recursions for 3472 fatgraphs—virtually impossible to conduct "by hand".

Chapter organization. In Section 3.2, 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 3.3). 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 3.4), using a number of indices equal to the treewidth. Figure 3.1 outlines the fundamental algorithm. Section 3.5, discusses extensions to combine multiple fatgraphs, include recursive substructure, and cover realistic energy models. Section 3.6 discusses the application of our methods to the design of concrete pseudoknot folding algorithms. We demonstrate the re-design of `gfold` for 1-structures, as well as the novel design of 2-structure prediction and interesting novel algorithms between 1- and 2-structures (e.g. predicting 5-chains in $O(n^7)$).

3.2 Definitions and main result

Notations and definitions. 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 $1, \ldots, n$), 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 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.

Fatgraphs and their expansions. A *fatgraph* [160, 16, 161, 162] 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 3.1 and Figure 3.2), 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.

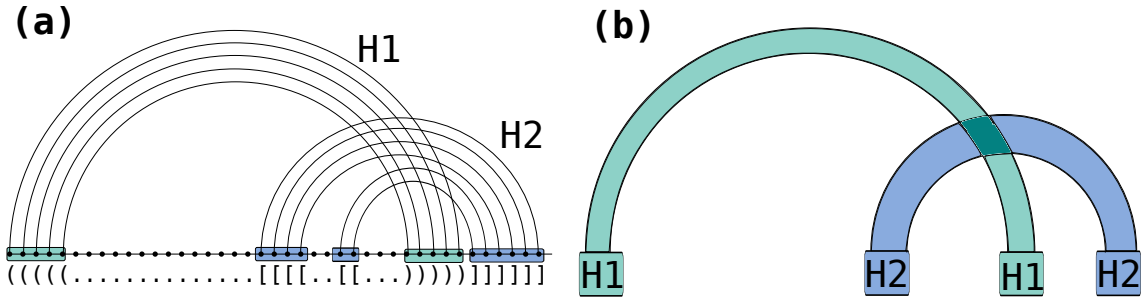


Figure 3.2: (a) Diagram of a secondary structure with two crossing helices (H1 green, H2 blue). (b) fatgraph corresponding to the above structure such that helices are collapsed into bands and form the shadow of the structure.

`gfold` connection. To make the connection to `gfold` [16] 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 [16] 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.

Problem definitions. In this chapter, 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 insertions. Only later, in Section 3.5, we extend to the full problem in realistic energy models.

Problem 10 (Fatgraph MFE folding).
Input: Collection of fatgraphs $\gamma_1, \ldots, \gamma_p$, sequence S
Output: Minimum Free Energy (MFE) arc-annotation for S according to a 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 20 (Fatgraph algorithm design problem).
Input: Collection of fatgraphs $\gamma_1, \ldots, \gamma_p$
Output: A Dynamic-Programming algorithm that, given any sequence S, solves the Fatgraph MFE folding problem over $\gamma_1, \ldots, \gamma_p$ and S.

Treewidth parameterization. Defining the treewidth of a fatgraph as the treewidth of its minimal expansion (see Section 3.3.2), our main result, stated in Algorithm 1, is the existence of an effective algorithm for the Fatgraph MFE-folding problem, parameterized by the maximum treewidth tw of the input fatgraphs. More precisely, it consists of an FPT preprocessing of the input fatgraphs, yielding an XP Dynamic-programming algorithm accepting any input sequence and solving the Fatgraph MFE folding problem (see Figure 3.1).

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: **for** each fatgraph γ_i **do**
2: Compute minimal expansion G_i of fatgraph γ_i ▷ Linear time; see Section 3.3.2
3: Find min. width tree decomposition $\mathcal{T}$ for G_i ▷ FPT in tw using exact tree dec. algorithm [77]
4: Transform $\mathcal{T}$ into a canonical form tree dec $\mathcal{T}'$ ▷ Polynomial time; see Section 3.4.1
5: Compute skeleton of $\mathcal{T}'$ ▷ Linear time; see Section 3.4.1
6: Derive corresponding DP scheme ▷ Linear time; see Section 3.4.2
7: **end for**
8: Run all DP schemes to find MFE arc-annotation of S ▷ XP in tw $O(n^{tw+1})$; See Section 3.5

Main result. The following result is the main result of our paper. A refined version is Theorem 16 in Section 3.4.3.

Theorem 13 (Main result). *Algorithm 1 solves the fatgraph folding problem in $O(n^{tw+1})$, where tw is the maximum treewidth of the input fatgraphs.*

As detailed with Theorem 16, the complexity can also be $O(n^{tw})$ in certain cases, depending on the choice of energy model and the fatgraphs under consideration. 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 3.7.

3.3 Minimal representative expansion of a fatgraph

Representative expansion: philosophy. 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 recursive 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 (and sometimes even $\mathcal{O}(n^{tw})$ for simple energy models).

3.3.1 Treewidth and tree decompositions

Let us recall the definition of tree decompositions (Definition 14 in Chapter 1, page 34)

Definition (Tree decomposition)**.** *Given a graph $G = (V, E)$, a tree decomposition $\mathcal{T}$ is a tuple $(T, \{X_t\}_{t \in T})$ with T a tree, and $\forall t \in T$, X_t a set of vertices of the graph, such that:*

1. *$\forall u \in V$, the set $T_u = \{t \in T \mid u \in X_t\}$ must be a connected subtree of T.*
2. *$\forall (u, v) \in E$, T_u and T_v must intersect.*

Notation 3 (vertex subtree)**.** $T_u = \{t \in T \mid u \in X_t\}$

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.

Reminder: treewidth and tree decompositions. Let us recall a few facts about treewidth, from Chapter 1, Section 1.2.3. Intuitively, the lower the treewidth, the closer G is to being a tree. Treewidth is NP-HARD to compute [76], but fixed-parameter tractable: there is a $O(f(w) \cdot n)$ algorithm [77] deciding whether $tw(G) \leq w$ given G. Many polynomial heuristics are also known to yield reasonable results [81], and optimized exact solvers have been developed [82, 119]. Notoriously, a wide variety of hard computational problems can be solved efficiently when restricted to graphs of bounded treewidth [165, 71], including in bioinfomatics [48, 166, 164]. Such is the case of pseudoknotted structure-sequence alignment, using the algorithm presented in [164]. The method presented in this paper can actually be seen as a generalization of this algorithm, allowing to perform "pseudoknotted *motif*-sequence alignment", with a motif describing a family of structures.

Minors and treewidth. 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 (Definition 17, page 39 and [84]), i.e. performing any sequence or edge contractions, edge deletions and vertex deletions on G can only lower its treewidth (Property 4, page 39). Second, degree-2 vertices can be contracted into their neighbors without changing the treewidth, as quickly stated below. This implies in particular that any bulge in a helix of an RNA structure graph is inconsequential with respect to treewidth.

Proposition 4. *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 $tw(G) = tw(G_{v \leftarrow u})$*

Proof. 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. □

Safe separators. Proposition 2 (page 36 in Chapter 1) established that the intersection of two adjacent bags in a tree decomposition is generally a *separator* (Definition 15) of the graph. Here, we import from [167] a converse result, essentially stating that turning a separator into a clique and recursively decomposing the components associated to it is a way to build (non-necessarily optimal) tree decompositions. It takes the form of an inequality valid for any separator S of G. The set of connected components obtained by removing S in G is denoted $\mathcal{C}_G(S)$.

Proposition 5. *If S is a separator of G, then*

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

with $G[C \cup \texttt{clique}(S)]$ 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 $G[C \cup \texttt{clique}(S)]$. 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. □

Minimal separators. To write down the proofs of the following section in a smoother fashion, we restrict (w.l.o.g) tree decompositions to be such that any intersection of two adjacent bags is a *minimal* separator of the graph.

Definition 21 (minimal separator)**.** A separator S of a graph G is *minimal* if $\forall u \in S$, $S \setminus \{u\}$ is not a separator of G.

In other words, "minimality" is meant with respect to inclusion. The existence of optimal decompositions such that all bag intersections are minimal separators is easily seen when defining tree decompositions in terms of *triangulations* and *chordal graphs* [82, 168]. In this framework, the treewidth of a graph G is the minimum possible maximum clique size in a chordal completion of G. The bags of the decomposition are the maximal cliques of the chordal completion ("clique-tree"), and intersections of adjacent bags are minimal separators. For completeness, we formulate this result in the following proposition:

Proposition 6. *Given a graph G, there always exists an optimal tree decomposition such that, for any two adjacent bags X and Y:*

1. *$X \cap Y$ is a minimal separator of G.*
2. *$|X \cap Y| \leq tw(G)$*

Proof. Denoting $\omega(H)$ the maximum clique size of a graph H, we have [168]:

$$tw(G) = \min_{H \text{ chordal completion of } G} \omega(H)$$

The tree decomposition corresponding to a particular chordal completion H of G is the "clique-tree" of H. Bag intersections are then minimal separators of G (item 1), and no two bags contain exactly the same vertices (hence item 2). We refer the reader to [168] for full definitions and justifications. □

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

Helices. Given an RNA graph (with one vertex per nucleotide and one edge per base pair and backbone link, see Figure 3.3 (a)), we call *perfect helix* a set of directly nested base pairs, resulting in the subgraph depicted on Figure 3.3 (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. Using the notations of Figure 3.3 (b), $\{u_1, v_1, u_\ell, v_\ell\}$ are called the *extremities* of the helix. Throughout the remainder of the article, helices will be often proven to be replaceable, as a subgraph, by one of two small graphs on 4 vertices. These two graphs are the clique on 4 vertices and a 4-cycle augmented with one (and only one) of the possible two chords. To simplify the exposition, we simply denote them by ⊠ and ◹.

Closing an helix with a clique. One situation where ⊠ will appear is when we prove that, sometimes, the 4 extremities can be connected into a clique without loss of generality. The graph we obtain, an helix closed by a clique, has treewidth 4, which will be an important threshold in our structural results below. We state this fact in the following lemma. Let us denote by H_l^* the graph corresponding to a helix of length l, with the extremities connected as a clique.

Lemma 2. *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 3.3 (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}. □

Main structural result. 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 3.3 (c)) .

Lemma 3. *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 5 that $tw(G) \leq \max(H \cup \texttt{clique}(S), G_{\boxtimes})$. The graph $H \cup \texttt{clique}(S)$ 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, per Lemma 2, 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 3.3 (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. □

Helix extensions and treewidth. Through the introduction of $G_{\boxtimes}$ and $G_{\boxslash}$ as the two possible graphs to which G is equivalent in terms of treewidth, Lemma 3 already contains the essence of Theorem 14, which will be the basis for generalizing tree decompositions of minimal expansions of a fatgraph to arbitrary helix lengths.

Theorem 14. *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.*

Proof. 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 no bag contains both an element from$\{u_i, v_i\}$ and $\{u_j, v_j\}$. I.e. the occurrences of $\{u_i, v_i\}$ and $\{u_j, v_j\}$

in the tree decomposition are completely separated by some edge (X, Y) of the tree decomposition. Indeed, if $\forall i, j \in [1, l]$ there is some edge between $\{u_i, v_i\}$ and $\{u_j, v_j\}$ 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, let us denote $S = X \cap Y$ the separator associated to that edge. By Proposition 6, 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 3.3(c), as well as on Figure 3.7. 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 occurrences 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 first look for a pair i, j verifying (as above) that some edge (X, Y) of the tree decomposition completely separates $\{u_i, v_i\}$ from $\{u_j, v_j\}$, although this time with no guarantee of finding one. If we do find one, we apply the same transformation as above.

In the case where no such pair i, j exists, we argue that the four extremities of the helix form a safe separator of G. i.e. $tw(G) = \max(4, tw(G_{\boxtimes}))$. An optimal tree decomposition for G can then be obtained from a tree decomposition $G_{\boxtimes}$, and a tree decomposition of an helix closed by a clique, connected through a bag in which the separator forms a clique. The helix can then simply be extended by changing the part of the tree decomposition representing the helix.

By Lemma 3, we have $tw(G) \leq \max(4, tw(G_{\boxtimes}))$. Since $tw(G) \geq 4$, it reduces to $tw(G) \leq tw(G_{\boxtimes})$. We now use the fact that edges connecting $\{u_i, v_i\}$ and $\{u_j, v_j\}$ for all i, j are represented in the tree decomposition to show that $G_{\boxtimes}$ is a minor of G, and therefore $tw(G) = tw(G_{\boxtimes})$

If there is an edge connecting u_i to v_j or v_i to u_j for $|j - i| > 1$ represented in the tree decomposition, then we obtain $G_{\boxtimes}$ through the contraction scheme represented on Figure 3.3. If $\forall i, j$ the edge connecting $\{u_i, v_i\}$ and $\{u_j, v_j\}$ is (u_i, u_j) or (v_i, v_j), then w.l.o.g we are in one of the two situations colored in orange on Figure 3.3. By contracting the orange parts into the extremity they contain, we get $G_{\boxtimes}$ as a minor of G. □

Minimal representative expansion. Since bulges in a helix only consist of vertices of degree exactly 2, combining Proposition 4 with Theorem 14 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 in between the extremities of an helix, arguments similar to the proof of Theorem 14 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 22 (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 3.8 (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. Interestingly, the two graphs $G_{\boxtimes}$ and $G_{\boxslash}$ that emerge in the proof of Lemma 3, as well as the separators they are associated to (see Figure 3.3 (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.

3.4 Interpreting the tree decomposition of a fatgraph expansion as a DP algorithm

Generating DP schemes: overview. 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 [16, 29].

3.4.1 Canonical form of fatgraphs tree decompositions

Canonical form: definition. Let us recall this 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.

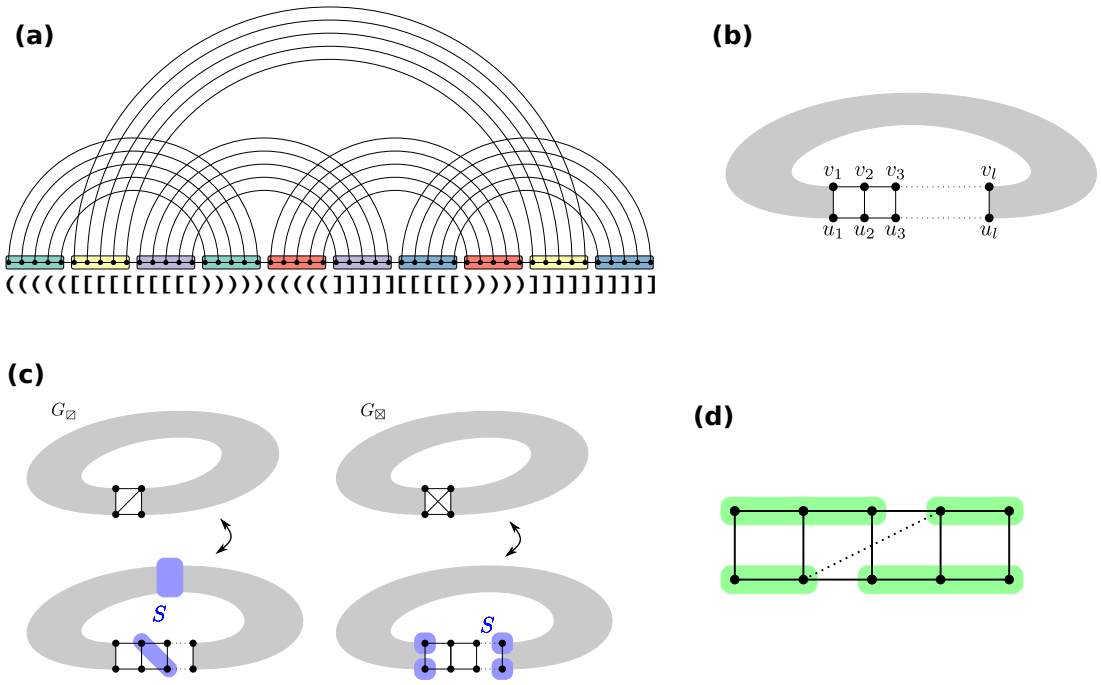


Figure 3.3: (a) minimal expansion of a fatgraph, with every helix of length 5, and no unpaired base. The associated graph consists of one vertex per base, and one edge per base pair and backbone link. (b) A helix of length l in an RNA graph, as per the latter definition. (c) Given a helix in a graph 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.

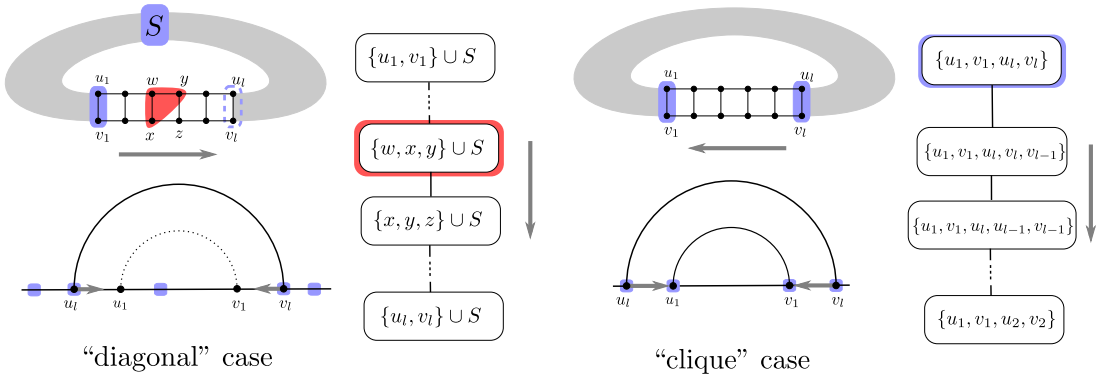


Figure 3.4: The two types of canonical representations for the helices of a graph completion G and associated dynamic programming schemes. (Left) In the Diagonal case, only the sequence positions of external (resp. internal) anchors are provided. Internal ones are obtained as the base case of an energy model-dependent dedicated dynamic programming scheme, propagating values for anchors in S along the way. (Right) In the clique case, all four anchors delimiting the helix have known position. Again, a dedicated dynamic programming algorithm is used to optimize over all possible contents for the helix, while accounting for associated free-energies.

Definition 23 (canonical form)**.** 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:** H is represented by a root bag that contains its 4 extremities , connected to a sub-tree-decomposition T_l recursively defined as

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

- **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

$$X_{2k+1} = S^* \cup \{v_{2k-1}, u_{2k}, v_{2k}\}.$$

The definition above is illustrated by Figure 3.4. A canonical tree decomposition for a minimum expansion of a fatgraph is also presented on Figure 3.5. It was obtained through the processing rou-

tine that we describe in Algorithm 2, applicable to any (optimal or not) tree decomposition. It can therefore use a sub-optimal tree decomposition obtained from a polynomial heuristic [165] instead of an exponential solver, if the latter is to time-consuming (although [82] is empirically quite efficient on RNA structure graphs).

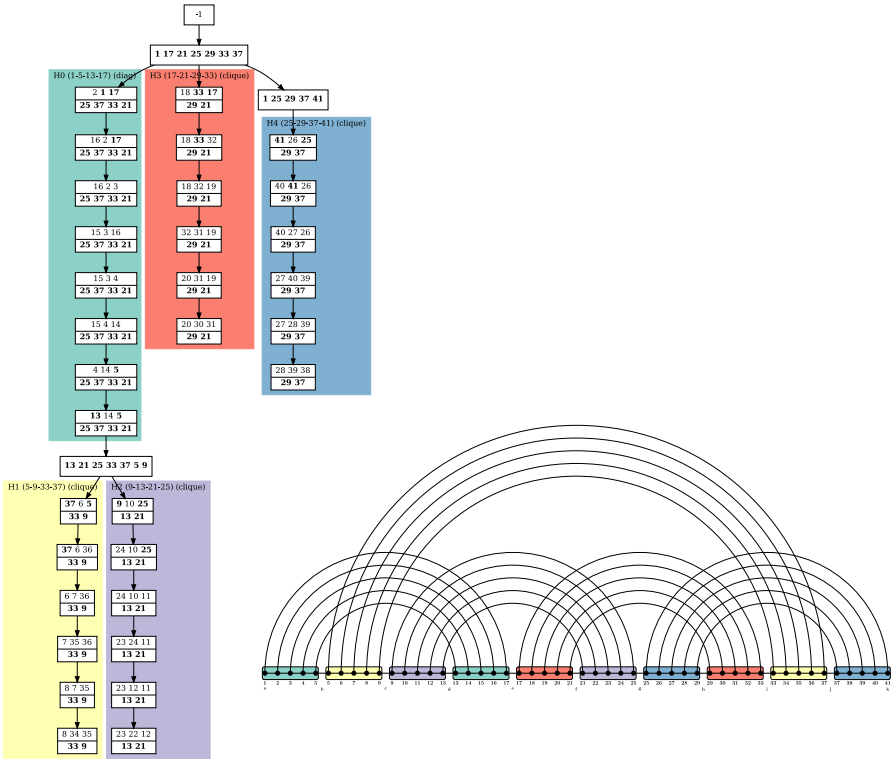


Figure 3.5: Canonical tree decomposition of the fatgraph given in Figure 3.1. White boxes represent the bags of the tree decomposition. Number in the bags correspond to the indices of the helices in the fatgraph where number on the bottom are kept while traversing the branch of the decomposition tree. Colored frames indicate the distinct helices (H0 to H4) of the structure. The tree decomposition was computed with the optimal solver [82], which we noticed is particularly efficient on RNA structure graphs.

Making tree decompositions canonical. Algorithm 2 essentially follows the dichotomy of the proof of Theorem 14. We state its correctness, run-time and proof below.

Theorem 15. *Given G the structure graph of a minimal expansion of a fatgraph γ, and $\mathcal{T}$ a tree decomposition of G, 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 in γ.*

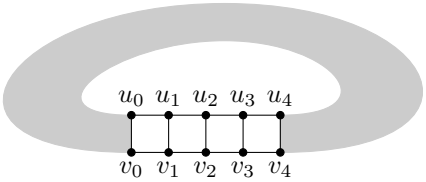


Figure 3.6: Sketch of an helix subgraph, in a minimal representative expansion of a fatgraph, along the annotation of vertices used in Algorithm 2. There is a slight abuse in using these same labels for each of the helices in the main for loop of Algorithm 2.

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. A representation of an helix as a subgraph in a minimal representative expansion, along with the notations (u_i, v_j...) used in this pseudo-code can be found on Figure 3.6. With a slight abuse of notation, we re-use these variables for each helix.

Input: Tree decomposition $\mathcal{T}$ for the minimal expansion G of a fatgraph γ.
Output: A tree decomposition of G in canonical form.

if width$(\mathcal{T}) \leq 3$ **then** ▷ "Diagonal case" only
  for helix H in fatgraph γ **do** ▷ $\exists i, j$ s.t. u_i, v_i completely separated from u_j, v_j in $\mathcal{T}$
    Find an edge (X, Y) of $\mathcal{T}$ and i, j such that $0 \leq i, j \leq 4$, $|i - j| > 1$
      and $X \cap Y$ separates u_i, v_i on the X-side from u_j, v_j on the Y-side
    $\forall i' \in [0 \dots 4]$, replace $u_{i'}$ with u_1 and $v_{i'}$ with v_1 in all bags of the X-side of $\mathcal{T}$
    $\forall j' \in [0 \dots 4]$, replace $u_{j'}$ with u_4 and $v_{j'}$ with v_4 in all bags of the Y-side of $\mathcal{T}$
    Insert between X and Y the "diagonal" canonical representation for H,
      with constant part $S = (X \cap Y) \setminus \{u_k, v_k\}_{i \leq k \leq j}$
  end for
else
  for helix H in fatgraph γ **do**
    if $\exists$ i, j and (X, Y) edge of $\mathcal{T}$ s.t $X \cap Y$ separates u_i, v_i on the X-side from u_j, v_j on the Y-side **then**
      ▷ "Diagonal case"
      $\forall i' \in [0 \dots 4]$, replace $u_{i'}$ with u_1 and $v_{i'}$ with v_1 in all bags of the X-side of $\mathcal{T}$
      $\forall j' \in [0 \dots 4]$, replace $u_{j'}$ with u_4 and $v_{j'}$ with v_4 in all bags of the Y-side of $\mathcal{T}$
      Insert between X and Y the "diagonal" representation for H,
        with constant part $S = (X \cap Y) \setminus \{u_k, v_k\}_{i \leq k \leq j}$
    else ▷ "Clique case"
      $\forall i, j$ there is always an edge connecting u_i, v_i to u_j, v_j represented $\mathcal{T}$
        $\rightarrow$ use these edges to get a tree decomposition for $G_{\boxtimes}$
      Attach a tree decomposition for an helix closed by a clique to the bag
        containing the clique on the 4 extremities of H
    end if
  end for
end if

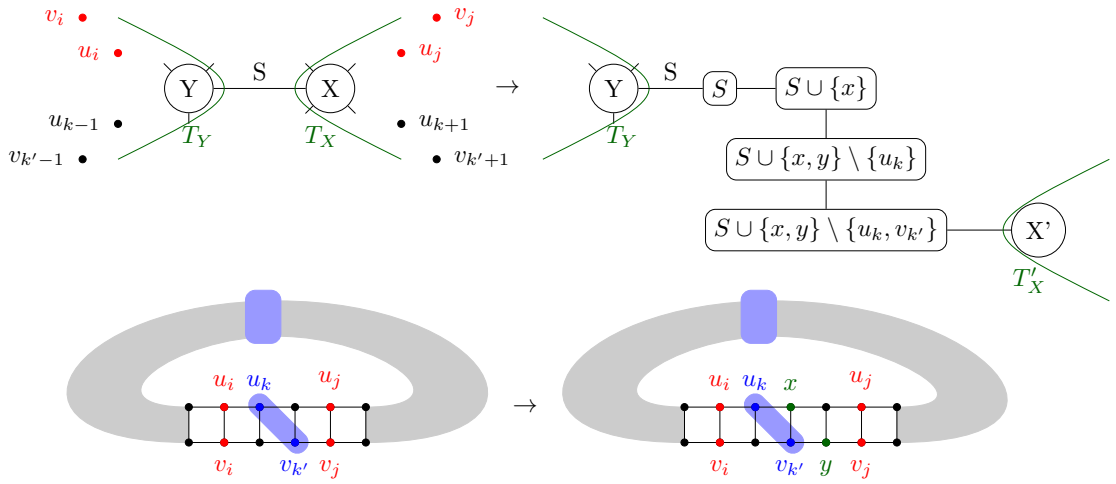


Figure 3.7: Representation of the local rewriting of a tree decomposition next to a separator S separating to base pairs (u_i, v_i) and (u_j, v_j), in order to extend a helix by one unit, through the introduction of new vertices x and y. This is used in Theorem 14, in what corresponds in Section 3.4 to the "diagonal" case.

Proof. 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: it essentially follows the dichotomy of Theorem 14. If width$(\mathcal{T}) \leq 3$, then there has to be a pair of indices i, j such that $\{u_i, v_i\}$ is separated from $\{u_j, v_j\}$ by an edge (X, Y) of the tree decomposition. If it is not the case, contracting (u_k, v_k) $\forall k$ yields a K_5-minor, which is not possible with a width of 3. We therefore get a separator as depicted in blue on Figure 3.7, which forms the "constant part" of the diagonal-case helix representation. The replacement of vertex occurrences on both sides of the separator does not increase the width, while representing all edges of the graph.

If width$(\mathcal{T}) \geq 4$, if a separator as above is found (but this time, no guarantee to find one), then we apply the same transformation. Otherwise, we use the extra edges represented in the tree decomposition to modify it into a tree decomposition of $G_{\boxtimes}$, as in the proof of Theorem 14. There is then necessarily a bag containing all four extremities of the helix, to which a tree decomposition representing the inside of the helix can be attached. □

Skeleton of a decomposition. 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 24 (skeleton)**.** 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 23.

Figure 3.8 (b) gives an example of such a skeleton.

3.4.2 Automatic derivation of dynamic programming equations in a base pair-based energy model

Deriving DP schemes. 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. We initially restrict our exposition to a base-pair based model, further named weighted-bps model, as defined in Section 1.1.3 (Definition 3, page 17). We recall that in this model, the free-energy of a structure S is given by:

$$E_{\texttt{w-bps}}(S) = \sum_{(i,j)\in S} w_{S[i]S[j]}$$

$w_{S[i]S[j]}$ being the contribution of a base-pair (i, j) to the free-energy (or negative log-odd to produce max-likelihood structures).

DP table definitions. Essentially, we introduce two DP tables for each helix, and one "transitional" tables for non-helix bags. The variables indexing these tables are called *anchors*. These integer variables 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 definition of a valid tree decomposition, a skeleton is guaranteed to:

1. Feature each anchor in some bag;
2. Represent each pair of consecutive anchors in at least one bag;
3. Propagate anchor values, such that the anchor values within helix tables remain consistent. This implies that 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.

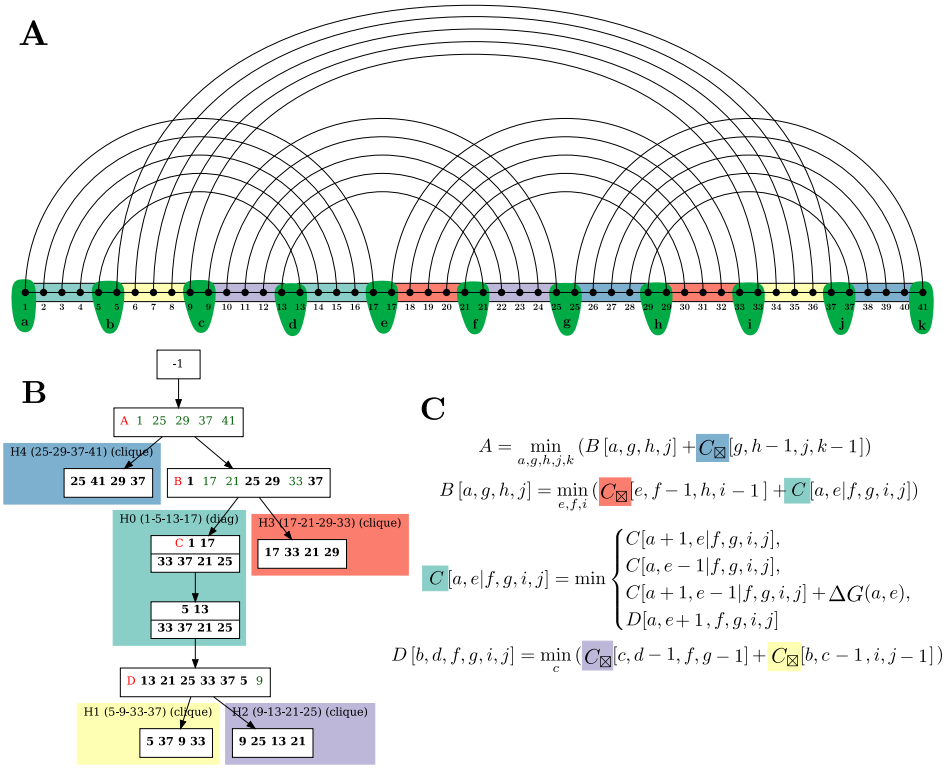
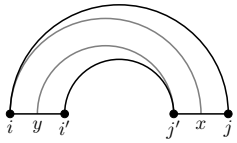


Figure 3.8: (A) Minimal representative length-5 expansion of the fatgraph shown in Figure 3.1. Anchor variables are highlighted in green. We introduce one such variable per gap between helices. (B) Skeleton of the tree decomposition. White boxes represent transitional bags, introducing/propagating indices, 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 3.4.2.



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.

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 3.8 (c) for an example of how anchor values are passed to $C_{\boxtimes}$ with a decrement of -1 for the same reason).

Recursive relations. 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:

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

and

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

where $w_{S[i]S[j]}$ denote the free-energy contribution of the base pair (i, j) in the input RNA sequence.

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 3.9.

Recursive relations. 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 commitment to form base pairs for neither i nor j (resp. where i is committed to form a pair with some position $x \leq j'$). We have:

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

and

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

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

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, i.e. anchors that are not found in the parent bag. Let I_P be the anchors of the parent bag of M in the tree decomposition, we have:

$$M[I_P] = \min_{\substack{\text{Values for} \\ \text{anch. in } I \setminus I_P}} \sum_{k=1}^{\#\text{child.}} \begin{cases} M_k[I_k] & \text{if } k\text{-th child trans.} \\ C'_{\boxtimes}[i, i'-1, j', j-1] & \text{if clique at } (i, i', j', j) \\ D'_{H_k}[i, j-1 \mid S_k] & \text{if diagonal at } (i, j') \end{cases}$$

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.

Automated C code generation. Figure 3.8 shows an example of output to our pipeline, with automatically generated LaTeX equations for the dynamic programming scheme inferred from the tree decomposition. Figure 3 gives other examples of such automatically generated equations. But our implementation, available freely at `https://gitlab.inria.fr/bmarchan/auto-dp`, is also capable of automatically generating C code implementing these equations. The automatically generated

`*.c` files corresponding to all of the examples of Figure 3 are available in the Supplementary Material of [100]. In the current state, they are only meant as a prototype demonstration. Developments towards generation of fully functional code, including the extensions presented in the next Section, will be the subject of future work.

3.4.3 Complexity analysis

Complexity of generated DP scheme in BP model. Let $w_{\boxtimes}$, $w_{\boxslash}$ and w' be the maximum width of a clique, diagonal and transitional bag (i.e. its size minus one; or 0 if no bag exist for a given type) in a canonical tree decomposition $\mathcal{T}$ of a fatgraph γ. Note that $w_{\boxtimes}$ is always 4, but we keep this notation for consistency. In the following theorem, γ is a fatgraph with $|\gamma|$ helices and $\mathcal{T}$ is a canonical tree decomposition for γ. The DP scheme obtained from $\mathcal{T}$ as described in the previous section is called the DP scheme *inferred from* $\mathcal{T}$.

Theorem 16. *In the BP energy model, the DP scheme inferred from $\mathcal{T}$ yields an algorithm for the Fatgraph MFE Folding problem with $O(|\gamma| \cdot n^{max(w_{\boxtimes},w_{\boxslash},w'+1)})$ time and $O(|\gamma| \cdot n^{max(w_{\boxtimes},w_{\boxslash},w')})$ space complexity.*

Proof. The complexity of the DP scheme inferred from $\mathcal{T}$ (presented in the previous section for a base-pair based model) depends on the complexities of filling each of the tables corresponding to helices. $C_{\boxtimes}[i,i',j',j]$ and $C'_{\boxtimes}[i,i',j',j]$ take $O(n^4)$ to fill, using either a memoization procedure or a bottom-up iteration of all possible values for i,i',j',j. It is equal to the space complexity thanks to the finite number of cases in their recursive equations. A similar analysis holds for $C_{\boxslash}[i,j \mid S]$ and $C'_{\boxslash}[i,j \mid S]$, except that the number of indices is $|S|+2$. Since the maximum size of a bag in a diagonal-case representation is $|S|+3$, we indeed have $w_{\boxslash} = |S|+2$. For transitional bags, the situation is slightly different. The indices of the table are the intersection with the parent bag in the tree decomposition, whose number is bounded by w'. The space complexity of the corresponding DP table is therefore $O(n^{tw'})$. But there is also a minimization over all possible values for the variables not present in the parent bag, incurring a linear factor for each of them. Overall, for a transitional B of maximum size $w'+1$, the complexity of filling the matrix is $O(w'+1)$ ($O(n^{|B\setminus P|})$ for each of the $O(n^{|B\cap P|})$) entries. As for the number of tables, it is at most twice the number of bags in $\mathcal{T}$, which is linear in the number of helices in γ. The overall time complexity is therefore given the DP table of most expensive filling cost, $O(|\gamma| \cdot n^{\max(w_{\boxtimes},w_{\boxslash},w'+1)})$. The same holds for the space complexity, yielding $O(|\gamma| \cdot n^{\max(w_{\boxtimes},w_{\boxslash},w')})$. □

Metric mismatch. Since tree decompositions are typically chosen to minimize their width $tw := max(w_{\boxtimes}, w_{\boxslash}, w')$, then the precise resulting complexity may depend on the choice of an optimal tree decompositions. Indeed, it could be that $tw = w'$, yielding a $O(n^{tw+1})$ algorithm or, conversely, $w' < tw - 1$ would imply a complexity of $O(n^{tw})$. In other words, in the base pair model, the algorithm induced by the choice of an arbitrary tree decomposition T may be suboptimal by a linear factor.

Exploring the space of tree decompositions. Fortunately, it is possible to work around this issue, and obtain a $O(n^{tw})$ DP algorithm anytime a suitable canonical fatgraph decomposition exists. To find such a decomposition, we explore the space of all possible canonical tree decompositions, through an enumeration of all possible representations for each helix. This is formalized in the theorem below (note that this is purely meant as a *feasibility* result, we do not expect this approach to be optimal in terms of complexity; indeed we conjecture that this subproblem is FPT for the treewidth of γ). We use the same notations as above by calling $w'(\mathcal{T})$ the maximum width of a transitional bag of a canonical tree decomposition.

Theorem 17. *Let G be a minimal expansion of a fatgraph γ with n_H helices. If there exists an optimal canonical tree decomposition $\mathcal{T}$ of G such that $w'(\mathcal{T}) \leq tw(G) - 1$, then such a $\mathcal{T}$ can be found in $2^{O(|\gamma|^2)} \cdot f(tw)$ time.*

Proof. The space of all possible canonical tree decomposition can be iterated over by deciding, for each helix, whether it is in the "clique" or "diagonal" case. If it is in the diagonal case, one must in addition decide what is the "constant part" of the representation of the helix. Any set S such that $\{u_1, v_1, u_5, v_5\} \cup S$ separates the graph into at least 3 connected components, one being the inside of the helix, is an eligible candidate. This process corresponds to deciding, for each helix, what separator cuts out the inside of the helix from the rest of the graph. When such a decision is made, a canonical tree decomposition can be obtained by computing canonical tree decompositions for the connected components associate to the separator, and connecting them together (in the spirit of Proposition 5). When there are no helices left, an optimal tree decomposition of the graph is computed in time $f(tw)$. It yields the transitional bags in between helix representations. Given that S is only composed of helix extremities, it is chosen among $\leq |\gamma|$ vertices. We consider therefore an upper bound of $2^{|\gamma|}$ for the number of possible choices of S in the diagonal case, and an upper bound of $|\gamma|$ for the number of connected components associated to a separator, the overall time of exploring all canonical tree decompositions is bounded by $O((|\gamma| \cdot 2^{|\gamma|})^{|\gamma|} \cdot f(tw)) \subseteq 2^{O(|\gamma|^2)} \cdot f(tw)$. If an optimal canonical tree decomposition $\mathcal{T}$ such that $w'(\mathcal{T}) \leq tw(G) - 1$ exists, then it corresponds to a particular assignation of separators to each helix as outlined above, and it will be one of the tree decompositions explored by the iteration. □

3.5 Extensions

Current limitations. 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. This enables us to solve the stated algorithm design problem (Def. 20) and consequently the associated folding problem in complex energy models, and discuss the consequences on the complexity.

3.5.1 More realistic energy models

Reminder: stacking and Turner models. For the sake of simplicity, we illustrated in Section 3.4.2 the generation of a dynamic programming algorithm within a fairly simple base-pair based energy model. However, the procedure can be adapted to capture more complex energy models found in the literature. This includes stacking base pairs models (Definition 4), which we recall are defined as:

$$E_{\texttt{stacking}}(\mathcal{A}) = \sum_{(i,j)\in\mathcal{A} \text{ s.t } (i+1,j-1)\in\mathcal{A}} w_{S[i]S[j],S[i+1]S[j-1]}$$

for a structure S, and with $w_{S[i]S[j],S[i+1]S[j-1]}$ the energy of base pair $(i+1, j-1)$ stacking onto (i, j).

Turner model. In the Turner model, also called nearest-neighbor model, any pseudoknot-free structure S is decomposed into loops. As defined in Chapter 1 (Definition 5), A loop is rooted at a base pair (i_0, j_0), and delimited by a set of base pairs $\{(i_k, j_k)\}_{1\leq k\leq p} \subset S$ such that $\forall k$, $(i_k, j_k) \subset (i_0, j_0)$ and $\forall k, k' \in [1 \dots p]$, $(i_k, j_k) \parallel (i_{k'}, j_{k'})$. A loop ℓ, with nucleotide content $c(\ell)$ is then assigned a free-energy contribution $w(c(\ell))$ that depends on the nucleotide content of base pairs, and unpaired regions between adjacent base pairs. The overall free energy of a structure in the Turner model is then defined as

Using Notation 1

$$E_{\texttt{Turner}}(S) = \sum_{\ell \text{ loop}} w(c(\ell))$$

Finally, we recall that the the Turner model usually uses affine linear models for multiloops ($p \geq 2$), and interior loops ($p = 1$), the latter based on loop length and asymmetry.

Adapting our framework for the stacking and Turner models. Both of those models can be captured by a modified versions of the dynamic programming algorithm presented in Section 3.4.2. In the stacking model, it suffices to duplicate the cliques (resp. diagonal) matrices to keep track of (i, j) being directly enclosed ($\perp$) or not ($\not\perp$) within a base pair $(i+1, j-1)$. This results in a replacement $(C_{\boxtimes}, C'_{\boxtimes})$ with $(C_{\boxtimes,\perp}, C'_{\boxtimes,\perp}, C_{\boxtimes,\not\perp}, C'_{\boxtimes,\not\perp})$ (resp. (D_H, D'_H) into $(D_{H,\perp}, D'_{H,\perp}, D_{H,\not\perp}, D'_{H,\not\perp})$), and the inclusion of suitable energy contributions for the $\perp$ cases, the only ones likely to form stacking pairs. The time complexity remains identical, up to a constant, to that of the BP energy model.

A consideration of the full Turner model is more involved, but can be achieved in $O(n^3)$ through an enumeration of all possible loops, as shown by Lyngsoe *et al* [169], by exploiting the linear interpolation of loops beyond a certain length threshold. Adapting the recurrence to consider all possible helix expansions of cliques and diagonals will result in a $O(n)$ time overhead for all cliques and diagonals, leading to an increased time complexity in $O(|\gamma| \cdot n^{\max(w_{\boxtimes}+1, w_{\boxdot}+1, w'+1)})$, or equivalently $O(|\gamma| \cdot n^{tw+1})$. A summary of the complexity of solving each type of recursive equation (clique, diag and transitional) depending on the energy model is given in Table 3.1.

Space complexity. Perhaps a more intuitive way of interpreting our treewidth-based framework is to look at the space complexity of the schemes it generates. It is the same regardless of the model, and directly dependent on the treewidth of the minimal expansion of the fatgraph under consideration, as stated below.

Lemma 4. *The space complexity of the generated DP schedule is $O(|\gamma|n^{tw})$, regardless of the energy model.*

Proof. The set of indices of a table is the intersection of the corresponding bag with its parent bag. Both bags have size at most $tw + 1$, and they are distinct, so their intersection has size at most tw. Each index runs in the range $[0, n]$, so the size of each table is at most n^{tw}. The number of tables is bounded by the number of bags in the tree decomposition of γ, which is itself in $O(|\gamma|)$. □

3.5.2 Integration with classic DP algorithms for MFE structure prediction

Disjunction over several fatgraphs. Firstly, let us note that alternative fatgraphs can easily be considered, without significant overhead, by adding a disjunctive rule at the top level of the DP scheme, such as

$$\text{MFE}_{\text{PK}} := \min_{i=1}^{p} \text{root}_{\gamma_i}[\emptyset]$$

where root_{γ_i} is the top level case 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.

Enriching classic schemes with fatgraphs. Fatgraphs usually represent a structural module rather than a complete RNA conformation. The role of the framework presented in this chapter is therefore primarily to supplement the classic DP scheme for 2D structure energy-minimization with additional constructs, enabling the consideration of pseudoknots. Towards that goal, 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 fatgraphs. In other words, one needs to be able to prescribe the span of the fatgraph occurrence, i.e. the values (i, j) of its extremal anchors (a, a') within the dynamic programming.

Adding a virtual overarching base-pair. To ensure this possibility, one simply needs to connect the first and last positions within the minimal fatgraph completion $G = (V, E)$, i.e. resulting in a graph $G' := (V, E \cup \{(a, a')\})$. Since each arc of the input graph is represented in a valid tree decomposition, we know that any tree decomposition for G' features a bag B including both a and a', possibly in conjunction with additional anchors $S := \{k_1, k_2, \ldots\}$. Moreover, since a tree decomposition is unordered, it can be re-rooted to start with B, and preceded by a root node restricted to anchors (a, b), without adverse consequences complexity-wise. This yields the following entry point for the DP of a fatgraph γ:

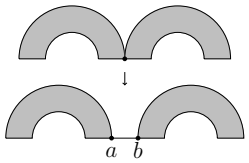
$$\text{MFE}_{\gamma}(i, j) := \min_{i<k_1<k_2<\ldots<j} M_B[i, k_1, k_2, \ldots, j]$$

Energy model	Diagonal tables $C_{\boxslash}[i,j\|S]$	Clique tables $C_{\boxtimes}[i,i',j',j]$	Transitional tables $M_X[I_X]$
BP-based model	$O\left(n^{\|S\|+2}\right)$	$O\left(n^4\right)$	$O\left(n^{\|I\|}\right)$
BP+stacking	$O\left(n^{\|S\|+2}\right)$	$O\left(n^4\right)$	|
Full Turner	$O\left(n^{\|S\|+3}\right)$	$O\left(n^5\right)$	|

Table 3.1: While the space complexity of the generated DP schemes is always bounded by $O(n^{tw})$ (Lemma 4), the run-time complexity of filling-up the DP tables $C_{\boxslash}$ and $C_{\boxtimes}$ depends on the choice of energy model. As for the table corresponding to a transitional bag X with indices I, the cost of filling it is $O(n^{tw+1})$ irrespectively of the energy model.

which can be used within a classic, pseudoknot-oblivious, DP schemes for MFE structure prediction. Complexity-wise, it can be shown that the additional base pair can at most increase by 1 the treewidth (and frequently leaves it unchanged).

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.



Splitting common helix anchors. 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 $\text{MFE}_{\text{SS}}(a+1, b-1)$.

Substructure insertion. To enable the insertion of substructures within a 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 $\text{MFE}_{\text{SS}}(i, k) + \delta$, where δ is the free-energy contributed by the rest of the helix (e.g., to include additional terms associated with multiloops).

3.5.3 Partition functions and ensemble applications

Uniqueness and unambiguity. For ensemble applications of our DP schemes, such as computing the partition function [170] and statistical sampling of the Boltzmann ensemble [171], it is imperative for the DP scheme above to be complete and unambiguous [172]. 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

			Complexities	
Type	Fatgraph	Treewidth	Full Turner	All others
H-type	`([)]`	4	$O\left(n^5\right)$	$O\left(n^4\right)$(*)
Kissing hairpins	`([)(])`	4	$O\left(n^5\right)$	$O\left(n^4\right)$
"L" [16]	`([{)]}`	5	$O\left(n^6\right)$	$O\left(n^6\right)$
"M" [16]	`([{)(]})`	5	$O\left(n^6\right)$	$O\left(n^6\right)$
4-clique	`([{<)]}>`	5	$O\left(n^6\right)$	$O\left(n^6\right)$
5-clique	`([{<A)]}>a`	5	$O\left(n^6\right)$	$O\left(n^6\right)$
5-chain	`({[)(][)}]`	6	$O\left(n^7\right)$	$O\left(n^7\right)$

Table 3.2: Table listing pseudoknot classes, corresponding treewidth and resulting complexity of the folding algorithm. For H-type pseudoknots beneath the Turner model, marked as (*), an iterated computation over canonical tree decompositions is required to achieve the complexity (see Theorem 17). For the H-type and kissing hairpins cases, 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 3. 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.

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 these two properties, we conclude that the partition function for a fatgraph (or several, possibly recursively and/or within a $\pm$ realistic energy model) can be obtained through the simple change of algebra pioneered by McCaskill [170] in the pseudoknot-free case. Namely, replace the $(\min, +, \Delta G)$ terms into $(\sum, \times, e^{\beta \Delta G})$, with $\beta = RT$ being the Boltzmann constant multiplied by some absolute temperature.

3.6 Automated (re-)design of algorithms for specific pseudoknot classes

Automated re-derivation of `gfold`. Our pipeline for automated generation of DP folding equations given a fatgraph has been implemented using `Python` and `Snakemake` [173]. The implementation is freely available at https://gitlab.inria.fr/bmarchan/auto-dp. Since the algorithms in [16] 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 3.2) 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 [16] 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 [174]). 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 [16] 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.

Other automated derivations of famous schemes. 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) [29], 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 3.2. Most remarkably, we design an algorithm optimizing over recursive expansions of kissing hairpins in $O(n^4)$, whereas CCJ [175, 45], 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 $O(n^8)$ [16]). 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 [30] to 6-dimensional instead of 4-dimensional DP matrix elements—both of which have never been implemented).

3.7 Conclusions and discussion

Overview. In this work, we provided 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.

Perspectives. In addition to the extensions described in Section 3.5, this work suggests perspectives that will be explored in future work.

Algebraic dynamic programming. 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 [176, 177, 174]. This would enable extensions to multiple context free grammars or tree grammars when describing the problem in the ADP framework.

RNA–RNA interactions. 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 [178]. More ambitiously, categories of pseudoknots inducing an infinite family of fatgraphs, e.g. as covered by the seminal Rivas & Eddy algorithm [30], 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 which would ensure the availability of the relevant anchors in some bags of the tree decomposition, allowing to score such non-contiguous, recursive substructures.

Optimality. 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 formally-defined 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. More precisely, given a fatgraph γ, one could imagine that a DP scheme (with DP tables indexed by *anchor variables* as is typically the case) capable of exploring all recursive expansions of γ would in particular induce a *decomposition* of the minimal representative expansion of γ, from the *parsing* of this structure by the DP grammar. If this decomposition can be reinterpreted as a tree decomposition, then the treewidth of the minimal expansion would become a lower bound on the number of indices to use in such a DP scheme.

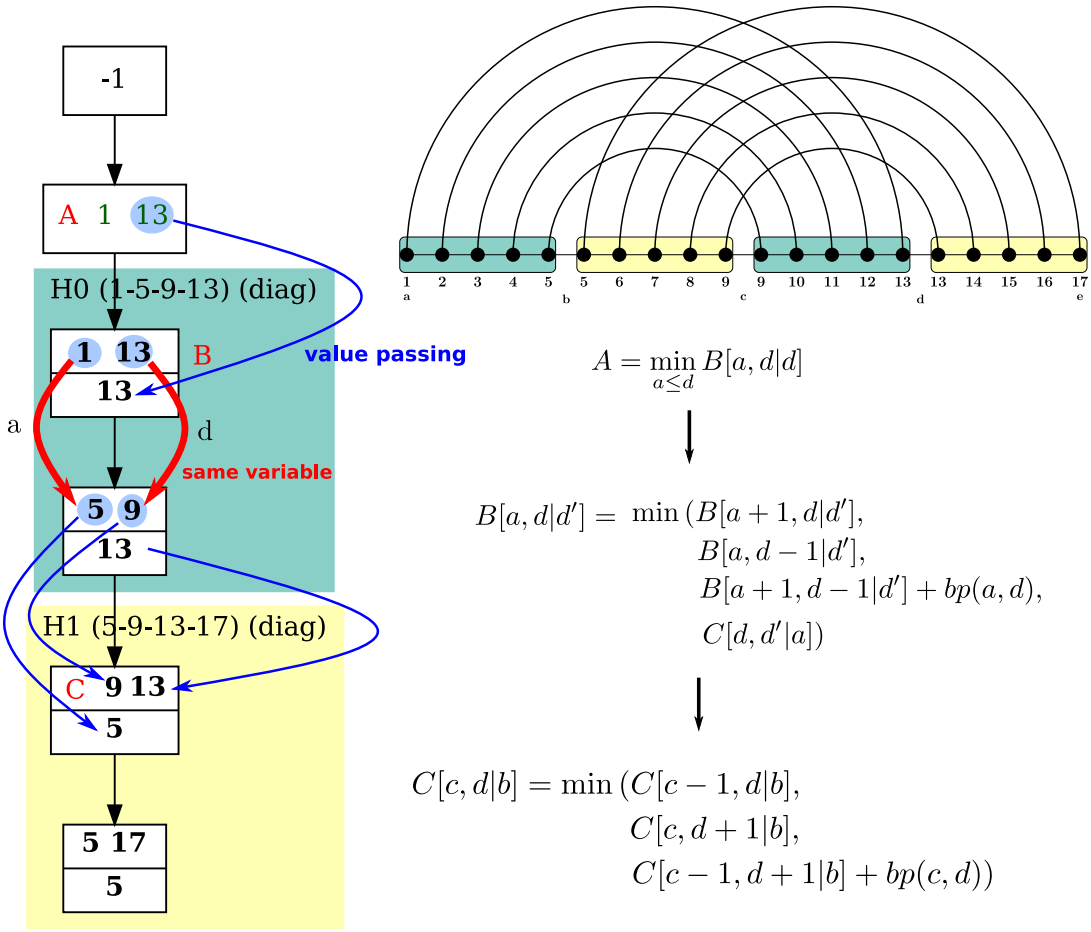


Figure 3.9: Derivation of DP equations from a skeleton, starting from the canonical tree decomposition of a length-5 expansion for a simple H-type fatgraph. On the left-hand-side, special emphasis is given to explaining how values are mapped at the end of a diagonal case. Extra tables $C'_{\boxtimes}$ and D'_H, needed to ensure unambiguity of the DP scheme, are omitted for the sake of simplicity without adverse consequences to correctness.

Chapter 4

Models and methods for pseudoknotted structure-sequence alignment

Abstract

When studying non-coding RNAs, we primarily expect *structure* conservation rather than strict sequence conservation. In other words, we expect to see sequence variability insofar that it conserves compatibility with a structure thought to guarantee the RNA's *function*. In particular, *compensatory mutations* on both ends of a base-pair are to be expected (e.g., GC→AU). Still, it could be the case that some portions of sequence are perfectly conserved across members of a non-coding RNA family.

Practical problems involving nc-RNAs may then consist in (1) building *consensus models* for homolog sequences, i.e. compact mathematical representations of the variability of sequences within a family, and (2) deciding whether a new sequence is part of a family, given a consensus model for the family (membership problem). The latter problem is akin to the Structure-Sequence Alignment problem mentioned in previous chapters. `InfeRNAl` [61] provides a concrete solution to these problems for pseudoknot-free secondary structures, using *covariance models* [179] as consensus models for ncRNA families. It has been used to build `Rfam`, a database counting more than ~ 4000 RNA families, and the central reference for ncRNA data. However, the problem of building consensus models for *pseudoknotted families*, with a fast enough solution to the membership problem, is still open.

After reviewing the theory behind `InfeRNAl`, this chapter presents experimental evaluation results for `LiCoRNA`, an implementation of [62] capable of taking pseudoknots into account. However, the consensus model of `LiCoRNA`, which simply consists in an arc-annotated secondary structure, misses some features with respect to `InfeRNAl`, namely *position-dependent statistics* and *stacking terms*. Therefore, we also present in this chapter a formulation of *pseudoknotted covariance models*. A salient feature is its use of a treewidth-based algorithm for the membership problem.

Contents

4.1 Introduction

Consensus models. In this chapter, we call *consensus model* any method capable of modeling the variability of sequences within a ncRNA family. By *ncRNA family*, we mean a set of sequences sharing a *consensus structure*, thought to be descendants of a common ancestor and to have a similar biological function. When building a consensus model for a ncRNA family, a typical starting point is a trusted multiple-sequence alignment of sequences in the family, along with secondary structure annotation, such as the one depicted on Figure 1.4 page 13 in Chapter 1. In the methods presented in this chapter, a typical choice will be the manually curated *seed alignments* of `Rfam`.

Covariance models [179] are the state-of-the-art consensus model solution for *pseudoknot-free* ncRNA families. They are implemented by `InfeRNAl`, and are used to build `Rfam`. Since the last section of this chapter consists in generalizing them to pseudoknots, we recall their definition below.

4.1.1 Covariance models

covariance models: definition. A covariance model [179, 61] is a statistical model of RNA sequence generation. It contains *nodes* arranged in a binary tree, the *guide tree*, which follows the tree of a conflict-free secondary structure. When representing a ncRNA family, this structure is the *consensus structure* of the family, thought to be adopted by all sequences in the family. An example of a covariance model, and the corresponding consensus secondary structure, is given on Figure 4.1.

There are 8 different nodes in a covariance model, as shown in the left column of Table 4.2. The most important ones are `MATP`, `MATL`, `MATR` and `BIF`, as they are in direct correspondence with the structural motifs of a consensus secondary structure. Each node contains a set of *states*, divided into a *main set* (outside of the brackets in Table 4.2) and a *split set* (inside the brackets in Table 4.2. Each state is connected to other states, in a directed way. All edges between states are between neighboring nodes of the guide tree. If u is the parent of v in the guide tree, with $M(u)$ the *main set* of states of u, $S(u)$ its *split set* and $M(v)$ the main set of v, then the set of connections between states contains:

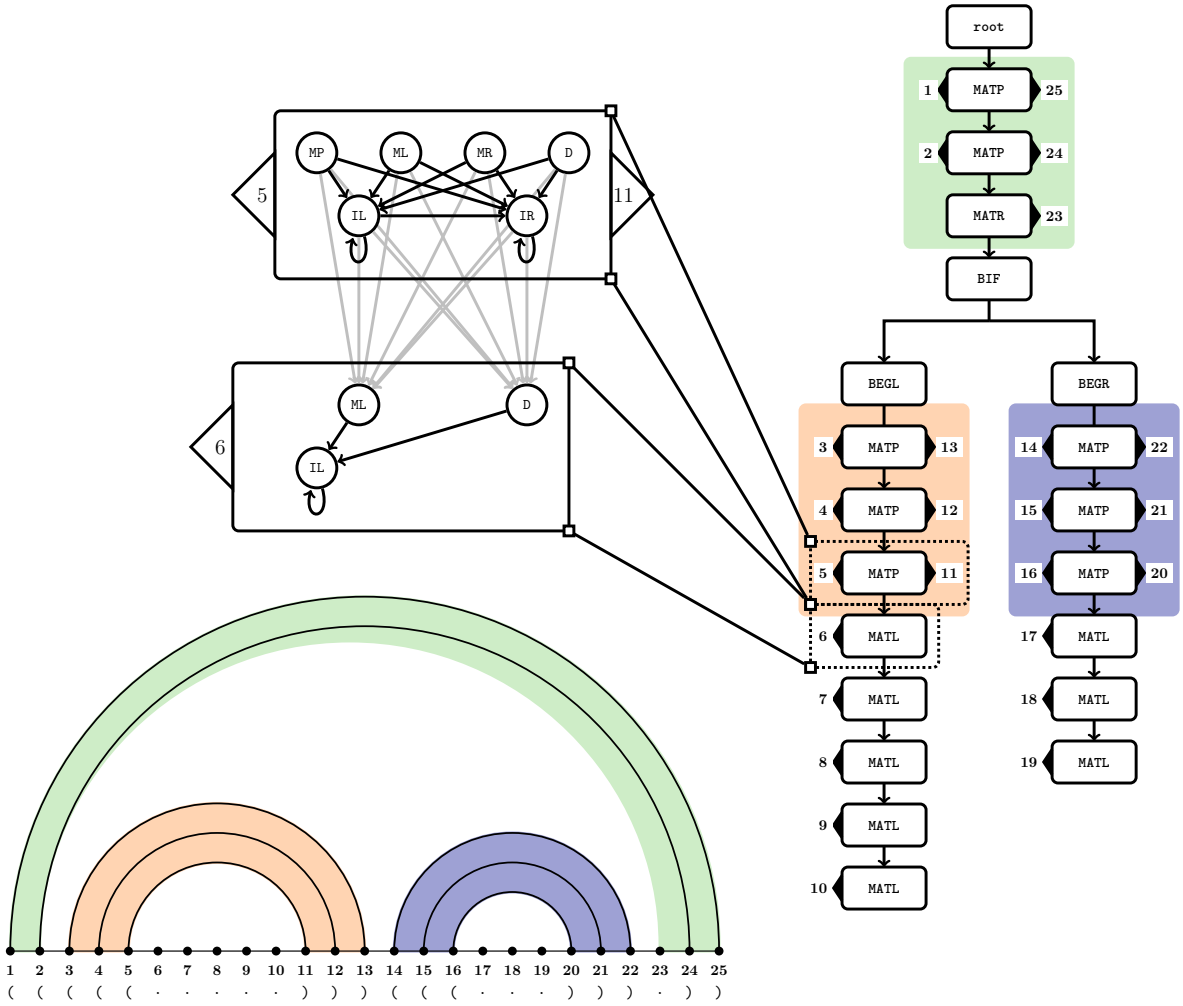


Figure 4.1: Sketch of the structure of an example of covariance model [179], with the corresponding secondary structure drawn on the bottom left. A covariance model is composed of nodes, corresponding to structural motifs in the secondary structure. These nodes contain states, corresponding to non-terminal symbols in the associated *stochastic context-free grammar*.

node	structural motif	states
MATP	base-pair	MP, ML, MR, D, [IL, IR]
MATL	left bulge	ML, D, [IL]
MATR	right bulge	MR, D, [IR]
BIF	multi-loop	B
ROOT	guide tree root	S, [IL, IR]
BEGL	stem start	S
BEGR	stem start	S, [IL]

Figure 4.2: Possible nodes in the guide of a covariance model, along with their state composition. States surrounded by [ ] constitute the *split set* of the node. The other states are its *main set*.

- $\{m \to s \mid m \in M(u) \text{ and } s \in S(u)\}$
- $\{\texttt{IL} \to \texttt{IR}\}$ within a node u such that $S(u)$ contains two elements IL and IR.
- $\{s \to m \mid s \in S(u) \text{ and } m \in M(v)\}$
- $\{m \to m' \mid m \in M(u) \text{ and } m' \in M(v)\}$

Examples of state connections between two different nodes are shown in Figure 4.1. Each directed edge between two states is weighted with a *transition probability*, usually in the form of a *log-odds score*. These probabilities are such that the sum of the out-going probabilities of a state sum to 1. The only exception to this rule is the bifurcation state B, which transitions with probability 1 to both of the S states (in BEGL and BEGR) it is connected to.

Sequence emission. Each state starting with M (MP,ML,MR) and I (IL,IR) is also associated with *symbol emission probabilities*. The overall picture is that, in a sequence of states starting at the root following the transition probabilities mentioned above, symbols are *emitted* whenever such a state is visited. Emitted symbols collectively form an RNA sequence, whose overall emission probability quantifies its likeliness to be a member of the ncRNA families. To make things more precise, let us describe the associated *grammar rules*.

Stochastic context-free grammar. Formally, a covariance model is a stochastic context-free grammar. Each of the state described above is associated to exactly one *non-terminal symbol*. The S state in a root node is the start symbol. Let us write down some of the production rules of such a grammar, starting with an MP state, associated with a base-pair (i, j):

$$\texttt{MP}_{ij} \xrightarrow[p(\texttt{MP}_{ij} \to V)]{p_{ij}(a,b)} a \cdot V \cdot b$$

With a and b terminal symbols, i.e. elements of {A,U,G,C}, emitted with probability $p_{ij}(a, b)$. V is one of the possible successors of $\texttt{MP}_{ij}$. For example, on Figure 4.1, the successors of $\texttt{MP}_{5,11}$ are

$\texttt{IL}_{5,11}, \texttt{IR}_{5,11}, \texttt{ML}_6$ and D_6. Symbol emission and transition to a successor state are independent, i.e. the emission of symbols a and b together with a transition to successor state V occurs with probability $p_{ij}(a,b) \cdot p(\texttt{MP}_{ij} \to V)$. The rules are similar for ML and MR states, with emission only to the left and right, respectively.

The production rule associated with a *bifurcation state* $\texttt{BIF}_{ij}$, corresponding to an interval $[i,j]$ in the consensus structure, with successor states $\texttt{S}_{ik}$ and $\texttt{S}_{kj}$, is the following:

$$\texttt{BIF}_{ij} = \texttt{S}_{ik} \cdot \texttt{S}_{kj}$$

While the production rule for a start state $\texttt{S}_{ij}$ is simply:

$$\texttt{S}_{ij} \xrightarrow[p(\texttt{S} \to v)]{} V$$

Finally, the production associated with an insert state $\texttt{IL}_k$ is:

$$\texttt{IL}_k \xrightarrow{p_{\texttt{IL}_k}(a)} a \cdot \texttt{IL}_k \mid V$$

For V a successor of $\texttt{IL}_k$. "Self-loop" transitions onto $\texttt{IL}_k$ occur with probability $p(\texttt{IL}_k \to \texttt{IL}_k)$. Alternately, transition to a successor state V occurs with probability $p(\texttt{IL}_k \to V)$. In any case, symbol emission of a occurs with probability $p_{\texttt{IL}_k}(a)$.

Parameter-learning. Starting from a multiple-sequence alignment with consensus structure annotation, we describe here how the parameters of a covariance model are learned. The overall strategy is a maximum-likelihood estimation, typically using a *Dirichlet prior* allowing for a simple estimation of probabilities from event counts. The first step is to build the *guide tree*. A simple rule to do so unambiguously is to consider that columns with $< 50\%$ of gap symbols contain symbols originating from *match states*, whereas columns with $\geq 50\%$ of gap symbols were emitted by *insert states*. The columns associated to match states, combined with the consensus secondary structure, yield the guide tree. Once the guide tree is built, each sequence in the alignment is an example of sequence of states, combined with symbol emissions at the match and insert states. A simple strategy for estimating symbol emission probabilities is then to use *Dirichlet priors* [180].

Maximum-likelihood parsing. Given a sequence S and a covariance model, membership in the ncRNA family is decided by the probability of the most-likely scenario (state transitions and symbol emissions) for the covariance model to generate S. The sequence of states then directly yields an *alignment* of the input sequence to the covariance model.

This score is computed through dynamic programming over the guide tree. Concretely, each entry $M[v,i,j]$ of the DP table contains the *maximum log-likelihood* that a sequence of states $\vec{s}$ starting at v generates $S[i:j]$. This likelihood can be expressed as:

$$\text{ll}(\vec{s}, S) = \log P(\vec{s}) + \log P(S \mid \vec{s})$$

For an MP state v, the recursive equation of M reads:

$$M[v,i,j] = \log P(v \text{ emits } S[i],S[j]) + \max_{w \in \mathtt{succ}(v)} [M[w,i+1,j-1] + \log p(v \to w)] \tag{4.1}$$

A similar equation holds for ML, MR, IL and IR states. For all the other states except B (i.e.S and D), the equation is also similar, but without the emission term. The equation for a bifurcation state (B) is the only one with a *marginalization* over some position. With b a bifurcation state, and S_1, S_2 its left and right successors, it reads:

$$M[b,i,j] = \max_{i \leq k \leq j} (M[S_1,i,k] + M[S_2,k,j])$$

RFAM [11]. `InfeRNAl` [61, 180], an implementation of the covariance models described above, is at the basis of the construction of `Rfam`. Starting from a manually-curated and trusted *seed alignment*, it builds a covariance model (command `cmbuild`). Then, the `rfamseq` [11] database is scanned for good *hits* of the covariance model, in the sense of the maximum-likelihood score defined above (command `cmsearch`). Finally, all hits and seed sequences are aligned with the covariance model, thanks to the backtrace of the dynamic programming scheme described above. This yields a *full alignment* for the entire family. (command `cmalign`). **The rest of this chapter is devoted to the design and benchmark of techniques capable of including pseudoknots into this pipeline**.

4.1.2 LiCoRNA

`LiCoRNA` is an implementation of the treewidth-based algorithm described in [62] for Structure-Sequence Alignment, and freely available at https://licorna.lri.fr/. In this subsection, we succinctly review both the definition of Structure-Sequence Alignment and the algorithm of [62]. Within this chapter, this problem is seen as the membership problem for a consensus model that simply consists of a pseudoknotted arc-annotated sequence.

Problem definition. As formalized into Problem 3, the task is to find a *monotonous mapping* (Definition 8) μ from an arc-annotated sequence $(Q, \mathcal{A})$ onto a plain sequence T, minimizing:

$$\mathtt{score}(\mu) = \sum_{(i,j)\in\mathcal{A}} \gamma_2(i,j,\mu(i),\mu(j)) + \sum_{i \text{ s.t } \mu(i)\neq\perp} \gamma_1(i,\mu(i)) + \sum_{\text{gap } g\subset Q} \lambda_Q|g| + c_Q + \sum_{\text{gap } g\subset T} \lambda_T|g| + c_Q$$

where γ_2 rewards base-pair conservation, and γ_1 rewards sequence conservation. A gap $g \subset Q$ is a consecutive set of positions in Q all sent to $\perp$ (deleted), with $|g|$ the number of deleted nucleotides. As for a gap $g \subset T$, it is the subsequence corresponding to the interval $]\mu(i) : \mu(i+1)[$ of T, when $\mu(i+1) > \mu(i)+1$. Note that *double gaps* are forbidden (if i and j are the bounds of a gap in Q, then $\mu(j) = \mu(i)+1$).

In order to present the algorithm of [62], we first define the notion of *structure graph*, which is central to its presentation.

Definition 25 (structure graph)**.** Given a set $\mathcal{A}$ of arcs over positions $[1 \dots N]$, the *structure graph* $G(\mathcal{A}) = (V, E)$ is the graph with vertex set $V = [1 \dots N]$ and edge set: $E = \mathcal{A} \cup \{(i, i+1) \mid i \in [1 \dots N-1]\}$

Treewidth-based algorithm. The algorithm presented in [62] uses the classic strategy of dynamic programming over a tree decomposition of the structure graph of $\mathcal{A}$. The recursive equations are quite similar to the example in Box 6, with a table c indexed by a bag identifier i and a *partial mapping* μ. This partial mapping is restricted to $P \cap X$, with X the bag identified by i and P its parent bag.

Precisions for optimal complexity. Without loss of generality, we work with *binary* tree decompositions. We also require a symmetric difference of at most one element between two adjacent bags. Both of these properties can be achieved from an arbitrary tree decomposition through bag duplications, without increasing the width [75].

We also adopt the convention that, instead of writing $\mu(i) = \perp$ (deletion), we use $\mu(i) = \mu(m)$ with m the largest position $< i$ such that $\mu(m) \neq \perp$. To guarantee the existence of m, virtual positions 0 and $|Q|+1$ are added left and right of Q, and likewise for T (0 and $|T|+1$), such that $\mu(0) = 0$ and $\mu(|Q|+1) = |T|+1$.

Recursive equations. In the following, μ^+ denotes a mapping on X (both $X \cap P$ and $X \setminus P$). The set of mappings on X agreeing with a partial mapping μ on $X \cap P$ is written $\text{comp}(\mu)$. Last, $Y_1 \dots, Y_\ell$ are the children bags of X, with identifiers $i_1, \dots, i_\ell$, and $\mu^+_{Y_j \cap X}$ is the restriction of μ^+ on $X \cap Y_j$.

With the precisions described above, $\ell = 2$ and $|X \setminus P| = 1$. As for the convention that if $i \in Q$ is deleted, then $\mu(i) = \mu(m)$, the mapped value of the last non-deleted position, it allows to easily compute $\text{comp}(\mu)$.

The recursive equation used by [62] is then:

$$c[i, \mu] = \min_{\mu^+ \in \text{comp}(\mu)} \left[\sum_{j=1}^{\ell} c[i_j, \mu^+_{Y_j \cap X}] + \texttt{lcost}(\mu^+) \right]$$

Where `lcost` must distribute the terms of the score function over the tree decomposition. Formally, `lcost` contains the following terms:

- $\gamma_2(i, j, \mu^+(i), \mu^+(j))$ for any $(i, j) \in \mathcal{A}$ such that X is the root of the subset of bags containing both i and j (which must be a tree)[1].
- when X is the closest to the root among the bags containing i:

[1] Following phylogenetics terminology, this root is often informally called the LCA of i and j, for lowest common ancestor.

- $\gamma_1(i, \mu^+(i))$ if $\mu^+(i) \neq \perp$
- λ_Q if $\mu(i) = \perp$

- and when X is the closest bag to the root containing both i and $i+1$:
  - and $\lambda_T(\mu^+(i+1) - \mu^+(i)) + c_T$ if $\mu(i+1) > \mu(i)$
  - c_Q if $\mu(i) \neq \perp$ while $\mu(i+1) = \perp$ (query gap opening)

Complexity analysis. The recursive equation above allows solving Structure-Sequence Alignment using dynamic programming. It yields an XP algorithm with respect to treewidth.

Proposition 7 ([62]). Structure-Sequence Alignment *can be solved in* $O(n \cdot m^{tw+1})$*, with* tw *the treewidth of the structure graph associated to* $\mathcal{A}$*,* $n = |Q|$ *and* $m = |T|$*.*

Proof. **Number of entries to** c: There are n possibilities of i and m^{tw} possibilities for μ (mapping $\leq tw$ positions into a set of cardinality m). This gives us a space complexity of $n \cdot m^{tw}$.
Time complexity: For each one of these entries, one must minimize over $\mu^+ \in \text{comp}(\mu)$. As there is at most one new vertex k when going from P to X, choosing μ^+ just consists in choosing $\mu(k)$ among $\leq m$ possibilities. Overall, the time complexity is indeed bounded by $O(n \cdot m^{tw+1})$. □

Features and limitations. The main feature of `LiCoRNA` is its ability to take pseudoknots and even non-canonical interactions[2] into account. Combined with the fact that natural RNA structure have reasonable treewidth (Figure 1.11, page 40), it shows that treewidth is a smooth and natural way to expand Structure-Sequence Alignment to the pseudoknot case.

Limitations. In our definition of Structure-Sequence Alignment, the score function does not take *stacking* into account, as the mapping of each base-pair is evaluated independently of the others. To include stacking, one would need, for two nested base-pairs (i, j) and $(i+1, j-1)$ in $\mathcal{A}$, to attribute an extra reward when *both* are conserved, i.e. mapped onto compatible nucleotides. In addition, recall that Structure-Sequence Alignment instances mainly come up in *ncRNA homolog search*, when scanning a window over a sequence database to identify new members of a given ncRNA family. In that context, a simple arc-annotated sequence $(Q, \mathcal{A})$ will, by design, fail to take into account the precise statistical sequence variability of the family. Both of these features are included in `InfeRNAl`, the current state of the art solution for pseudoknot-free ncRNA homolog search. These limitations therefore need to be addressed into formulate a true pseudoknotted extension of it.

[2] Non-canonical interactions are just additional arcs in $\mathcal{A}$. They might increase the treewidth, but otherwise seamlessly integrate into the framework.

Rest of this chapter. In spite of the limitations of `LiCoRNA`, we present in the next section a numerical evaluation of its capacity to create high quality alignments for pseudoknotted ncRNA families. The purpose is to get a quantitative sense of what is missed when not taking pseudoknots into account. Afterwards, and to conclude the chapter, we give a possible formulation for *pseudoknotted covariance models*. It uses a treewidth-based algorithm to align sequences onto the states of a *multiple-context free grammar* [181, 16].

4.2 Evaluating the quality of a pseudoknotted structure-sequence alignment methods

4.2.1 Evaluation methodology

Purpose and overview. The purpose of this numerical evaluation is to assess the capability of `LiCoRNA` to create better quality alignments than `InfeRNAl` for pseudoknotted ncRNA families. The experiments were run on data from pseudoknotted `Rfam` families. It consisted in evaluating several tools for structure-aware alignment for two different tasks: realigning seed sequences to see if the seed alignment is recovered (Experiment 1) and mapping arc-annotated seed sequences onto full sequences (Experiment 2). More details, including which *metrics* the different tools were evaluated upon, are given below.

selection of RFAM families and tools. `Rfam` [11] is the reference database for non-coding RNAs. It regroups sequences into *families* of homologs. For each of these families, a manually-curated *seed alignment*, with consensus secondary structure annotation, is given. It is used as a basis for building a *covariance model*, which is used (through `InfeRNAl`) to find candidate members of the family and build a *full alignment* out of them.

Family selection. Our study focused on `Rfam` families with pseudoknotted consensus structures. Among these pseudoknotted families, we selected those with at least 10 sequences in the seed alignment.

Alignment tools. In addition to `LiCoRNA` and `InfeRNAl`, we will also look at the performances of:

- `LoCaRNA` [43], a tool capable of *simultaneous alignment and folding* from a set of unaligned, plain sequences.
- The Smith-Waterman algorithm [182], and more precisely its `Rust-bio` [183] implementation.

We will also refer below to `Rfam` as an "alignment method". When we do so, we mean that we simply take the seed alignment and reduce it to an alignment of the sequences we are interested in.

Experiments. Starting from pseudoknotted ncRNA families in `Rfam`, we conduct the two following experiments. One of them consists in realigning the seed alignments of each family with various methods including `LiCoRNA`, to see if we recover the same quality as the manually curated versions available on `Rfam`. The other consists in aligning structure-annotated seed sequences with plain sequences from the full set. To give more details:

- (Experiment 1) **realigning seed alignments**. For each pseudoknotted family, pairs of seed sequences are aligned using one of them as query (annotated by the consensus structure) and the other as target. 15 sequences were selected in each family with `cd-hit` [184], which selects a set of representative sequences from an input alignment. Therefore, Experiment 1 consisted in computing $15 \times 14 = 210$ pair-wise alignments with each methods.
- (Experiment 2) **aligning seed sequences into full sequences** For each pseudoknotted family, seed sequences are used as queries (annotated by the consensus structure) and full sequences as targets. As for Experiment 1, 15 seed sequences and 15 full sequences were selected using `cd-hit` [184], resulting in 225 pair-wise alignments to compute for each family and each alignment method.

Computing environment and restrictions. Numerical experiments were launched on the Core Cluster of the Institut Français de Bioinformatique (IFB) (ANR-11-INBS-0013). The most time consuming tool to run was undoubtedly `LiCoRNA`. In fact, the results presented in this section are restricted to `Rfam` families for which more than 80% of the pair-wise `LiCoRNA` alignments were computed given ~ 3 weeks of computation on 300 cores of the IFB cluster, and a 6 giga-bytes memory limit for each individual alignment computation. In addition, to get more reasonable run-times, the maximum insertion gap length (i.e., the max possible value of $\mu(i+1) - \mu(i)$) when aligning Q and T with `LiCoRNA` was bounded to $\max(4, ||Q| - |T||)$.

Metrics. When aligning an arc-annotated sequence $(Q, \mathcal{A})$ onto a plain sequence T, we implicitly *predict* a structure on T, composed of the base-pairs of $\mathcal{A}$ mapped onto compatible nucleotides in T. In the case of Experiment 1, the seed alignment consensus structure gives us a "ground truth" structure for each target T. This allows us to use classic statistical metrics for evaluating alignment tools, namely sensitivity and positive predictive value. We denote $CS(T)$ the consensus structure restricted to base-pairs with nucleotides compatible in T.

Sensitivity and PPV. In statistics, sensitivity is the proportion of true positives in cases predicted as positives. In our case, it is the proportion of predicted base-pairs that are true base-pairs of the target, as per the consensus structure:

$$\text{sensitivity} = \frac{\text{\# correctly predicted bps}}{|CS(T)|}$$

with

$$\text{predicted bps} = \{(i,j) \in \mathcal{A} \mid (\mu(i), \mu(j)) \in \{(\texttt{A},\texttt{U}), (\texttt{G},\texttt{U}), (\texttt{G},\texttt{C})\}\}$$

and correctly predicted bps $=$ {predicted bps} $\cap\, CS(T)$. As for the positive predictive value (PPV), it is the proportion of cases predicted as positives that are true positives. In other words:

$$\text{PPV} = \frac{\text{\#correctly predicted bps}}{\text{\#predicted bps}}$$

AFI. Contrary to the quantities above, the Average Fractional Identity (AFI) does not stem from statistics. It compares two alignments, with the rationale that we will compare the alignments computed by the different methods to the `Rfam` reference. Formally, it is the proportion of identical *columns* in the two alignments. If the two sequences being aligned are S_1 and S_2, an identical column is when character number i of S_1 is aligned with character j of S_2 in both alignments. It is also called the sum-of-pairs-scores (SPS) in the literature [185]. In our experiments, it was computed using the implementation provided by [186].

BPCI. When aligning an arc-annotated sequence $(Q, \mathcal{A})$ onto a plain sequence T, the base-pair conservation index (BPCI) is the proportions of base-pairs of $\mathcal{A}$ that are mapped onto compatible nucleotides in T. If μ is the mapping corresponding to the alignment:

$$\text{BPCI}(\mathcal{A}, T, \mu) = \frac{1}{|\mathcal{A}|} \times |\{(i,j) \in \mathcal{A} \mid (\mu(i), \mu(j)) \in \{\texttt{AU},\texttt{UA},\texttt{GC},\texttt{CG},\texttt{GU},\texttt{UG}\}\}|$$

Given an arc-annotated sequence $(Q, \mathcal{A})$ aligned with a plain sequence T, we distinguish three different BPCI values, corresponding to different categories of base-pairs in $\mathcal{A}$. Indeed, given that `Rfam` annotates pseudoknots with letters (`A-a,B-b`...) on a "base" conflict-free structure, we distinguish pseudoknotted base-pairs from non-pseudoknotted base-pairs using this criteria. This gives rise to BPCI_{PK} and $\text{BPCI}_{\text{no-PK}}$, the proportions of conserved pseudoknotted (resp. non-pseudoknotted) base-pairs when mapping $(Q, \mathcal{A})$ onto T. As one might expect, when `InfeRNAl` is used to build the full `Rfam` alignment, the "pseudoknotted base-pairs" defined above are exactly the base-pairs that are not taken into account.

Nucleotide match. Finally, the "nucleotide match" metric is a pure measure of sequence conservation. Given an alignment of two sequences S_1 and S_2, it is simply the proportion of aligned symbols $S_1[i]$ and $S_2[j]$ such that $S_1[i] = S_2[j]$.

4.2.2 Results

Experiment 1 results interpretation. The results of Experiment 1 are summarized on Table 4.1. A graphical representation (scatter plots) is also given on Figure 4 and Figure 5 in Appendix C. We list below what we identify as take-away messages for these results. The overall picture is contrasted, and suggests that to truly evaluate the importance of taking pseudoknots into account in homolog sequence alignment and search, other experiments may be needed.

Experiment 1	AFI	sensitivity	PPV	nucl. match	BPCI	$\text{BPCI}_{\text{NO-PK}}$	BPCI_{PK}
`LiCoRNA`	0.74 ± 0.19	0.72 ± 0.19	0.82 ± 0.18	0.62 ± 0.12	0.90 ± 0.06	0.91 ± 0.06	0.85 ± 0.16
`LocARNA`	0.79 ± 0.12	0.70 ± 0.16	0.86 ± 0.12	0.59 ± 0.13	0.83 ± 0.10	0.84 ± 0.09	0.79 ± 0.20
`Rfam`	1.00 ± 0.00	0.92 ± 0.05	1.00 ± 0.00	0.55 ± 0.15	0.89 ± 0.07	0.88 ± 0.07	0.92 ± 0.14
`Smith-Waterman`	0.83 ± 0.09	0.56 ± 0.21	0.73 ± 0.19	0.66 ± 0.10	0.73 ± 0.15	0.72 ± 0.15	0.80 ± 0.19

Table 4.1: Summary of the results of **Experiment 1**. A full graphical representation of these results is given in Appendix C, on Figure 4. Values are the average obtained over the selected families with each method, plus or minus the standard deviation. For each metric, the method obtaining the best result is highlighted in gray. For AFI and PPV, `Rfam` (i.e., simply taking the seed alignment) yields a value of 1 by definition. For AFI, the second highest value of 0.83 for `Smith-Waterman` is remarkable, as it means that a pure sequence-based method yields an alignment that is more similar to the seed alignment than structure-based approaches. Another surprising result is the fact that, whereas `LiCoRNA` is the best method in terms of overall BPCI and $\text{BPCI}_{\text{NO-PK}}$, it is not as good for pseudoknotted base-pairs. A potential explanation for both these facts would be that, in `Rfam`, pseudoknots are primarily aligned based on **sequence similarity**. A more thorough analysis of the results is provided in the main text.

1. Except for BPCI values, `LiCoRNA` **and** `LocARNA` **have similar performances** on all other metrics. This is surprising because `LocARNA` does not take pseudoknots into account. A tentative explanation would be that, by taking into the *base-pairing probabilities* (Box 2, page 22 of Chapter 1), it may indirectly account for pseudoknotted base-pairs.

2. Apart from `Rfam`, which by definition has an AFI value of 1, **the** `Smith-Waterman` **alignments are the closest to the seed alignments** per the AFI metric. This points towards a high amount of sequence-based aligning in the seed alignments of pseudoknotted families in `Rfam`. One possible explanation is that, since `InfeRNAl` cannot take pseudoknots into account, the only way it may achieve some form of pseudoknot conservation is through strong sequence conservation. It may then be that, through iterations of the manual curation of seed alignments, such a pattern emerged as the most performing.

3. In terms of proportions of conserved base-pairs, i.e. BPCI values, `LiCoRNA` **conserves overall as much (and even slightly more) base-pairs as the seed alignment**. However, this surprisingly **stops being true when restricting the analysis to pseudoknotted base-pairs** (BPCI_{PK}). This means that optimizing for the score function of `LiCoRNA` does not yield the seed alignment of `Rfam`. A possible interpretation is that, when manually-curating seed alignments in `Rfam`, special care is taken to yield alignments compatible with pseudoknots. It would appear here that in fact, when globally optimizing for base-pair conservation (pseudoknotted and non-pseudoknotted alike) and sequence mapping, pseudoknots of the consensus structure for these families are not as robust as expected.

Experiment 2 results interpretation. The results of Experiment 2 are summarized on Table 4.2. Full graphical representations (scatter plots) of these results can be found on Figures 6, 7, 8 and 9 in Appendix C. We comment here on two salient aspects of these results.

Experiment 2	nucl. match	BPCI	$\text{BPCI}_{\text{NO-PK}}$	BPCI_{PK}
`LiCoRNA`	0.58 ± 0.11	0.92 ± 0.06	0.93 ± 0.06	0.84 ± 0.15
`LocARNA`	0.55 ± 0.13	0.83 ± 0.13	0.84 ± 0.12	0.79 ± 0.17
`InfeRNAl`	0.67 ± 0.10	0.90 ± 0.11	0.90 ± 0.11	0.90 ± 0.17
`Smith-Waterman`	0.48 ± 0.10	0.43 ± 0.17	0.41 ± 0.17	0.49 ± 0.22

Table 4.2: Summary of the results of **Experiment 2**. A full graphical representation is given on Figures 6,7,8 and 9 in Appendix C. As Table 4.1, the values are average scores of the selected set of `Rfam` families, along with their standard deviation , for each tool. Surprisingly, `LiCoRNA` achieves the highest BPCI values except when restricting the scope to pseudoknotted base-pairs. Another unexpected result is that `InfeRNAl` achieves a higher nucleotide match score than `Smith-Waterman`. A common tentative explanation to both of these facts would be that, when it comes to pseudoknotted base-pairs, the manually-curated seed alignment and consensus structure annotation are built based mostly on sequence conservation. When optimizing instead for a combination of both sequence and structure conservation, these base-pairs do not prove as robust as non-pseudoknotted ones.

1. `LiCoRNA` **obtains the best BPCI scores, except for pseudoknotted base-pairs**. The situation is similar to that of Table 4.1. We may therefore put forward the same tentative explanation, namely that pseudoknotted base-pairs in `Rfam` are annotated based mostly on sequence conservation. They turn out to be less robust then non-pseudoknotted base when optimizing for a combination of structure and sequence conservation.

2. `InfeRNAl` **obtains a higher nucleotide match score than** `Smith-Waterman`. This high nucleotide conservation score is consistent with the point made above, that `InfeRNAl` primarily uses sequence conservation to align pseudoknots. In doing so, it is more likely to conserve pseudoknotted base-pairs that have been annotated *in accordance with* this sequence conservation.

Conclusion: a contrasted picture. Our results do not clearly highlight the added value of taking pseudoknots into account when aligning putative members of pseudoknotted ncRNA families. Part of the difficulty may stem from the fact that, even though seed alignments are manually curated, they are still influenced by the inability of `InfeRNAl` to take pseudoknots into account. Indeed, our results point toward a qualitative *annotation difference* between pseudoknotted and non-pseudoknotted base-pairs. It seems that pseudoknot annotation is strongly based on *sequence conservation*. Then when using the seed alignment as a basis for a covariance model, and running `InfeRNAl` to build the full alignment, pseudoknots are conserved only as a result of strong sequence conservation.

Pseudoknotted covariance models. A more complete set of experiments, that might be able to highlight better the importance of taking pseudoknots into account, should allow for the *tweaking* of the seed alignment and its consensus structure. In combination, ideally, one would use a *pseudoknotted generalization* of infernal, which should allow to build "full sets" of sequences of better quality for pseudoknotted ncRNAs. In the latter, "quality" could be for instance measured in terms of increased covariation for pseudoknotted base-pairs, and higher overall statistical likelihood of family members

compared to other sequences.

The next section gives a tentative formulation of such *pseudoknotted covariance models*. It consists in generalizing the *stochastic context-free grammars* underlying `InfeRNAl` with *multiple context-free grammars* [181], and using a `LiCoRNA`-like treewidth-based algorithm for aligning sequences onto them.

4.3 Formulation of pseudoknotted covariance models

4.3.1 Rewriting `InfeRNAl` and `LiCoRNA` differently

We start by writing the score functions of `InfeRNAl` and `LiCoRNA` into a common framework. This will allow to write down a score function corresponding to a pseudoknotted version of covariance models. Interpretations in terms of stochastic grammar, and the question of maximum likelihood inference with a treewidth-based algorithm, will be addressed later sections.

Re-writing `InfeRNAl` differently. Sequence emission in `InfeRNAl` is the combination of the sampling of a *state sequence* and *symbol emission* at the states of that sequence equipped with emission probabilities (match and insert states). When aligning a sequence S with the covariance model (with equations such as Equation 4.1), we ask for the sequence of states $\vec{s}$ that maximizes:

$$\texttt{score}_{\texttt{InfeRNAl}}(\vec{s}) = \log P(\vec{s}) + \log P(\vec{s} \text{ emits } S)$$

with, if $\vec{s} = [s_0 \dots s_t]$, $\log P(\vec{s}) = \sum_{i=0}^{t-1} \log P(s_i \to s_{i+1})$. As for the probability that $\vec{s}$ emits S, let us define $\mu(i, \vec{s})$ in order to express it. With i a covariance model position (e.g., the i in a state like $\texttt{MP}_{ij}$), $\mu(i, \vec{s})$ is the position in the sequence S that i is aligned with, depending on the state sequence $\vec{s}$ (in particular the amount of *insert states* it has). With this notation:

$$\log P(\vec{s} \text{ emits } S) = \sum_{i \text{ s.t. } s_i = \texttt{MP}_{kl}} \log P(\texttt{MP}_{kl} \text{ emits } S[\mu(k, \vec{s})], S[\mu(l, \vec{s})]) + \sum_{i \text{ s.t. } s_i = \texttt{ML}_k, \texttt{IL}_k \text{ or } \texttt{MR}_k} \log P(s_i \text{ emits } S[\mu(k, \vec{s})])$$

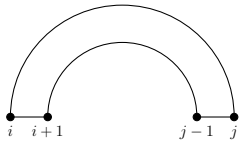
New variables. For any position i of the covariance model, let us introduce $\delta(i)$, a binary variable indicated whether i is *deleted* ($\delta(i) = 1$) or not $\delta(i) = 0$. In addition, for two consecutive positions $i, i+1$, we introduce $\eta(i, i+1)$, a binary variable indicating whether an insertion has taken place between i and $i+1$.

Notation 4. *δ (deletion) and η (presence of gap)*

Notations. Recall that a covariance model is a tree of nodes, with each node containing interconnected states. Given a node a, we write $\Delta(a)$ the set of variables $\delta(i)$, for any position i belonging to the node. For instance $\Delta(\text{MATP}_{ij}) = \{\delta(i), \delta(j)\}$. Likewise, we write $I(a)$ (for "insert") for the set

of η variables in the *split set* of a, i.e. corresponding to the insert states belonging to a. In addition, the tree of nodes (guide tree) is called T, with edge set $E(T)$. With these notations, the `InfeRNAl` score simply reads:

$$\texttt{score}_{\texttt{InfeRNAl}}(\vec{s}) = \sum_{a \to b \in E(T)} sc(\Delta(a), I(a), \Delta(b)) + \log P(\vec{s} \text{ emits } S)$$



with sc a score contribution consisting of transition probabilities within the states of a and into the states of b. Examples of how state sequences correspond to assignments of δ and η variables is given on Figure 4.3 For two stacked base-pairs (i, j) and $(i+1, j-1)$, it reads:

$$sc\left(\delta(i), \delta(j), \delta(i+1), \delta(j-1), \eta(i, i+1), \eta(j-1, j)\right)$$

Score contributions per helix. One way to define "helices" in `InfeRNAl` is to say that they are the pieces found between start, end, and bifurcation nodes. These nodes (`BEGL`, `BEGR`, `BIF`) only consist of single states (e.g., `B` for a bifurcation node). The transition between a `BIF` node and a "begin" node for another helix is deterministic. This architecture cuts off any scoring dependencies across helices, so that $\texttt{score}_{\texttt{InfeRNAl}}$ may also be seen as a sum of scores over helices. With $\mathcal{H}$ the set of helices, and each helix $H \in \mathcal{H}$ seen as a path of nodes:

$$\texttt{score}_{\texttt{InfeRNAl}}(\vec{s}) = \sum_{H \in \mathcal{H}} \left[\sum_{a \to b \in E(H)} sc(\Delta(a), I(a), \Delta(b)) + \log P(\vec{s} \cap H \text{ emits } S[H, \vec{s}]) \right] \quad (4.2)$$

Where $S[H, \vec{s}]$ is the portion of sequence aligned with helix H, given $\vec{s}$. The only interaction between helices is the fact of choosing "which parts of the sequence S" aligns with H, which needs to be done in a consistent way across helices. In the notations originally used for `LiCoRNA`, it corresponds to guessing μ. The consistency then corresponds to μ being *monotonous* ($\mu(i) \leq \mu(i')$ for $i < i'$).

$\vec{s}$: state sequence

Rewriting `LiCoRNA` differently. Let us recall the cost function that `LiCoRNA` optimizes for:

$$\texttt{score}(\mu) = \sum_{(i,j) \in \mathcal{A}} \gamma_2(i, j, \mu(i), \mu(j)) + \sum_{i \text{ s.t } \mu(i) \neq \perp} \gamma_1(i, \mu(i)) + \sum_{\text{gap } g \subset Q} \lambda_Q |g| + c_Q + \sum_{\text{gap } g \subset T} \lambda_T |g| + c_Q$$

where $\mu : [1 \dots |Q|] \to [1 \dots |T|] \cup \{\perp\}$ and $\mu(i) = \perp$ means that position i is *deleted*. By restricting μ to non-deleted positions, and using the variables δ and η to signal deletions and gaps, we can write:

$$\begin{aligned} \texttt{score}_{\texttt{LiCoRNA}}(\mu, \eta, \delta) = \sum_{i,j \in \mathcal{A}} sc_{\text{bp}}(\delta(i), \delta(j)) &+ \sum_{i \text{ s.t. } (\delta(i), \delta(i+1)) = (0,1)} c_Q \\ &+ \sum_{i \text{ s.t. } \eta(i, i+1) = 1} \lambda_T \cdot (\mu(i+1) - \mu(i)) + c_T \\ &+ \sum_{i \text{ s.t. } \delta(i) = 1} \lambda_Q + sc_{\text{seq}}(\mu, S) \end{aligned}$$

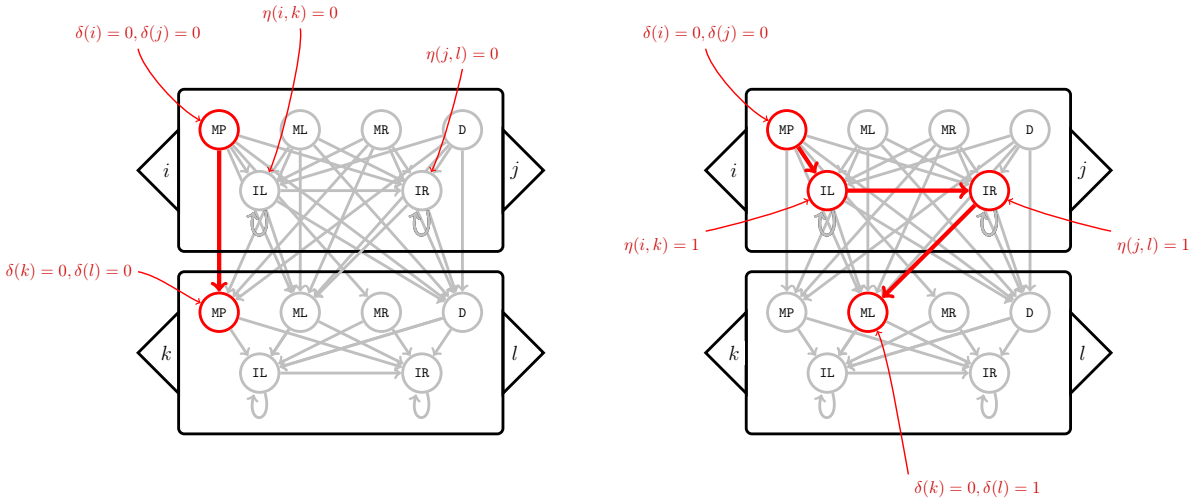


Figure 4.3: A sequence of states in a covariance may be specified by binary variables δ and η, associated to each position of the consensus structure. $\delta(i)$ means "is position i deleted?" while $\eta(i,k)$ means "is there a gap between i and k?". In the example shown above, $k = i+1$ and $l = j-1$, but separate letters were kept for clarity. The score function of both `InfeRNAl` and `LiCoRNA` may be expressed in terms of these variables, allowing for the development of a pseudoknot-aware covariance model (or a stacking-aware `LiCoRNA`, depending how you look at it).

With sc_{bp} accounting for the deletion (partial or complete) of base-pairs, and $sc_{\text{seq}}(\mu, S)$ accounting for any sequence-related term, i.e. γ_1 and some γ_2 terms from the original formulation.

4.3.2 Pseudoknotted covariance models

Overview of the structure. To describe the overall structure of our formulation of pseudoknotted covariance models, we first describe a way to partition a structure graph $G(\mathcal{A})$ into *helices*, when $\mathcal{A}$ is an arbitrary pseudoknotted secondary structure. Each base-pair and each unpaired position will belong to one helix, and the overall score function will be, as for `InfeRNAl`, a sum over helix-wise terms.

Partition into helices. Given $\mathcal{A}$ a (non-empty) pseudoknotted structure and $G(\mathcal{A})$ its structure graph (Definition 25), we first partition the base-pairs, unpaired positions, and "backbone edges" between positions into helices $\{H_1, \ldots, H_p\}$ as follows. Let bp_1, bp_2 be two base-pairs of $\mathcal{A}$, and x an unpaired position of $\mathcal{A}$.

- If $\text{bp}_1 \subset \text{bp}_2$ or $\text{bp}_2 \subset \text{bp}_1$ and they are in conflict with the same sub-set of $\mathcal{A}$, then they belong to the same helix. This relationship defines equivalence classes, which are starting point for helices.

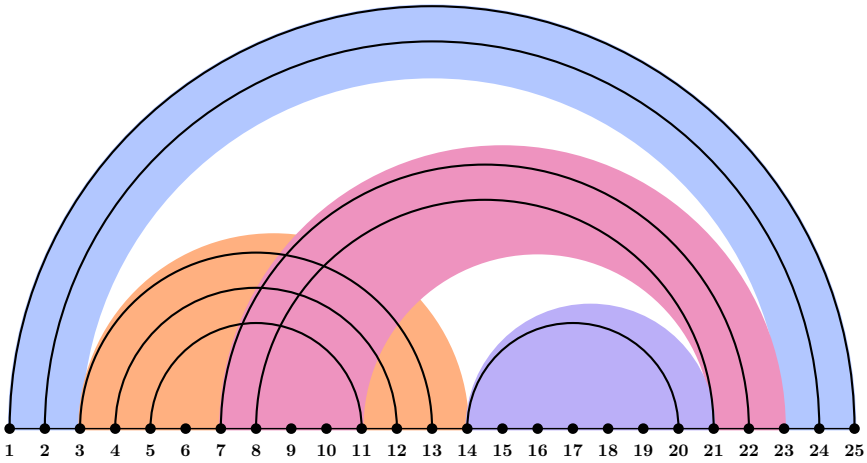


Figure 4.4: Example of a partition of a pseudoknotted structure into helices, so that unpaired positions and edges between consecutive positions are also assigned to an helix. For base-pair, the equivalence relation that defines these helices is to be (1) nested and (2) crossing the same set of other base-pairs. The score function of pseudoknotted covariance models will then be expressed as a sum of terms corresponding to each helix.

- Let y and z be the closest paired positions of x in $G(\mathcal{A})$, such that $y < x < z$, and bp_y, bp_z the corresponding base-pairs. If y (resp. z) does not exist, then x is assigned to the helix containing bp_z (resp. bp_y). If bp_y and bp_z belong to the same helix, then x also belongs to that helix. If they do not belong to the same helix, then we simply choose to assign it to the helix containing bp_y.
- Finally, we assign the "backbone edges" between consecutive positions to helices. If two neighboring positions belong to the same helix, then so does the edge connecting them. If they do not, we arbitrarily assign the edge to the left position.

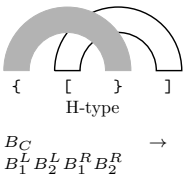
An example of such a partition of a pseudoknotted structure into helices is given on Figure 4.4. The core idea of our formulation of pseudoknotted covariance models is that **these helices are treated exactly like in** `InfeRNAl`. In particular, each of them is associated with a "directed path of nodes", starting with a "begin" node and followed by a succession of `MATP`, `MATL` or `MATR`. These nodes contain the same states as in `InfeRNAl`, connected in the same way.

Conflict-connected components. The description of the *multiple context-free grammar* associated with such crossing sets of helices requires defining conflict-connected components. Quite simply, two helices are in the same component if they cross, i.e. if the base-pairs they contain cross. In this decomposition, helices that do not cross any other correspond to a context-free grammar, just like in `InfeRNAl`. Only the non-trivial connected components need new grammar rules. Such a decomposition is illustrated on Figure 4.5. The components themselves form a tree, with conflicted components informally corresponding to multiple "parallel" helices. This tree is a generalization of the *guide tree*

of standard covariance model. We call it the *generalized guide tree.*

Associated stochastic grammar. Given a pseudoknotted secondary structure $\mathcal{A}$, and its decomposition into helices and conflicted-components described above, we give here a description of the multiple context-free grammar associated to it.

To start with, non-conflicted helices just correspond to standard context free grammar rules. We give here a description of rules for conflicted-components. Let C be such a component, containing helices $H_1, \dots, H_p$. Recall that C is simply seen as a "big node" in the generalized guide tree. Its grammar therefore simply starts with a "begin" non-terminal symbol B_C. Then, to each helix H_i is associated a pair of symbols B_i^L, B_i^R. To B_C is associated one deterministic production rule producing a succession of B_i^L, B_i^R symbols *following the conflict pattern of* $H_1 \dots H_p$. An example is given in the margin. Then, for each helix H_i, we have the rule:



$$(B_i^L, B_i^R) \to (A_i^L S_i^L, S_i^R A_i^R)$$

A_i^L and A_i^R are "anchor variables" that will be used once all of the helices in C have been "generated". They mark the potential "holes" of the component, in which sub-structures will be inserted (for example, on Figure 4.5, the start symbol of the purple sub-structure, starting with base-pair 14-20, will be connected to one of these "anchor variables" of the orange structure). As for S_i^L and S_i^R, they are simply a regular "start state" `S` that has been split in a left and right part. All states from the standard covariance model representation are split this way. For instance, the rule applying to an `MP` state is:

$$\left(\texttt{MP}_{ij}^L, \texttt{MP}_{ij}^R\right) \xrightarrow[p(\texttt{MP}_{ij} \to V)]{p_{ij}(a,b)} \left(a \cdot V^L, V^R \cdot b\right)$$

A similar split applies to all rules of the stochastic context-free grammar associated to a covariance model. Finally, when the "end" symbol of each helix is reached, some of the anchor variables may be used as a starting point for sub-structures. On the example given on Figure 4.5, the purple helix grammar rules would start from A_H^R, with H the 7-8/21-22 helix. If a "hole" is bordered by two of these anchors, we arbitrarily associate the starting of the substructure to the left one.

Terminology. For a pseudoknotted secondary structure $\mathcal{A}$, the set of nodes, states and grammar rules described above, combined with transition and emission probabilities, is a *pseudoknotted covariance model* associated to $\mathcal{A}$.

Score function. Given a pseudoknotted covariance model associated to $\mathcal{A}$, and a sequence S, we use the variables μ, δ, η defined in the previous subsections to explore the set of state successions and emissions producing S. The score function on μ, δ, η is then simply the same as `InfeRNAl`. With $\mathcal{H}$ the set of helices in $\mathcal{A}$, and each helix $H \in \mathcal{H}$ seen as a path of nodes:

$$\texttt{score}_{\texttt{pkcm}}(\mu, \delta, \eta) = \sum_{H \in \mathcal{H}} \left[\sum_{a \to b \in E(H)} sc(\Delta(a), I(a), \Delta(b)) + \log P(\vec{s} \cap H \text{ emits } S[H, \vec{s}]) \right] \quad (4.3)$$

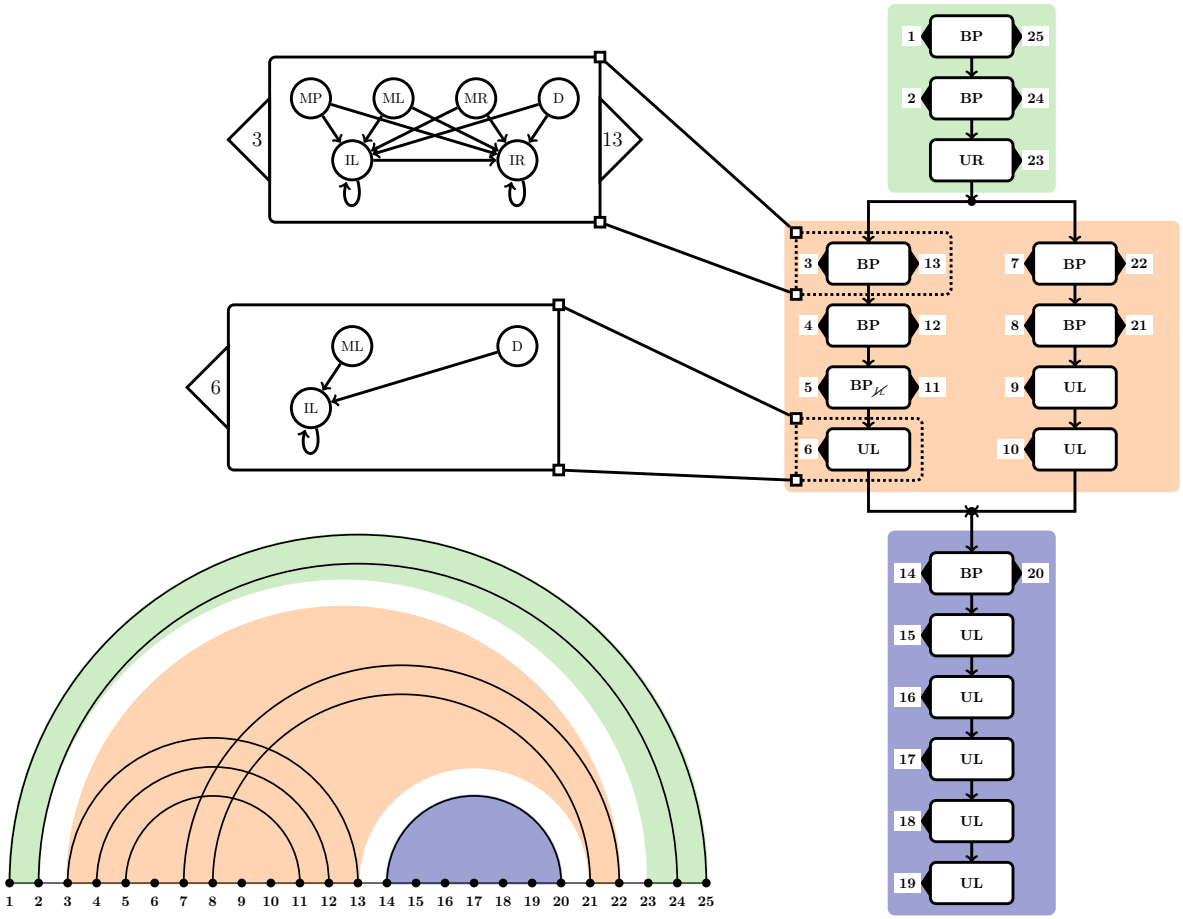


Figure 4.5: Example of the structure of a pseudoknotted covariance model. Each helix is represented with a succession of `InfeRNAl`-like nodes (`MATP`,`MATL`,`MATR`), containing the same states, and connected the same way. Conflicted helices, here depicted "in parallel" require grammar rules acting on couples of symbols, making the corresponding grammar a multiple context-free grammar [181, 16]. The score function of a pseudoknotted covariance model is essentially the same as the one of `InfeRNAl` (Equation 4.2). The originality is the `LiCoRNA`-like treewidth-based algorithm for the alignment problem.

With $\vec{s}$ a set of states, specified by μ, δ, η, and $S[H, \vec{s}]$ the part of the sequence aligned with H given μ, δ, η. Recall that $\Delta(a)$ and $I(a)$ are respectively the set of δ and η variables associated to node a.

To flesh it out a little bit more, let us see the terms involved for two perfectly stacked base-pairs i, j and k, l ($k = i + 1$ and $l = j - 1$). We write the overall contribution $sc^{i,j,k,l}_{\text{stack}}$:

$$sc^{i,j,k,l}_{\text{stack}} = sc_{\text{seq}}(S, \delta(i), \delta(j), \mu(i), \mu(j)) + sc_{\vec{s}}(\delta(i), \delta(j), \eta(i,k), \eta(l,j), \delta(k), \delta(l)) \\ + sc_{\text{gap}}(\mu(i), \mu(k), \eta(i,k)) + sc_{\text{gap}}(\mu(j), \mu(l), \eta(j,l))$$

With sc_{seq} that depends on which state among $\texttt{MP}, \texttt{ML}, \texttt{MR}, \texttt{D}$ is selected for (i, j), and how likely it is that the corresponding characters in S (at positions $\mu(i), \mu(j)$) are then generated. The (binary) variables $\delta(i), \delta(j)$ specify which of the four states is chosen. $sc_{\vec{s}}$ scores the likeliness of transitioning among the states of the 2 base-pairs, and the insert states in between. A pictorial representation of such terms, for all possible transitions (up to symmetries) between $\texttt{MATP}, \texttt{MATL}$ and $\texttt{MATR}$ nodes is given on Table 4.3. In the "variable graph" column, each μ, δ, η variable corresponds to a vertex, and two vertices are connected if they appear together in a scoring term.

4.3.3 Aligning sequences to a pseudoknotted covariance model

Augmented structure graph. Aligning sequences onto a pseudoknotted covariance model (PKCM) will be done through dynamic programming over a tree decomposition. Compared to `LiCoRNA`, which works with a tree decomposition over the *structure graph* $G(\mathcal{A})$ of the consensus structure $\mathcal{A}$, aligning a PKCM involves a tree decomposition of an *augmented* structure graph $G^+(\mathcal{A})$.

Where $G(\mathcal{A})$ can be seen as a graph with one vertex per $\mu(i)$ variable, for i any consensus position, $G^+(\mathcal{A})$ takes as vertices all μ, δ and η variables. Two variables are connected if they need to be scored together by a term in the cost function $\text{score}_{\texttt{pkcm}}$. This graph construction is akin to *cost function networks* [187] or *constraint networks* [188], i.e. networks of variables in which edges represent dependence.

Another way of constructing it, given a pseudoknotted secondary structure $\mathcal{A}$, is to first decompose $\mathcal{A}$ into helices $H_1, \ldots, H_p$. In turn, each helix H is decomposed into structural elements (base pairs, left bulge and right bulge) following an `InfeRNAl`-like representation ($\texttt{MATP}$, $\texttt{MATL}$ or $\texttt{MATR}$). The transitions between each of these elements are replaced the corresponding sub-graphs on the right column of Figure 4.3. The resulting graph $G^+(\mathcal{A})$ contains $G(\mathcal{A})$ as a sub-graph, when selecting μ variables only.

Distinguishing vertices: heavy and light. When guessing the optimal assignment of (μ, δ, η), each $\mu(i)$ has a domain of possible values of size m with $m = |S|$, the size of the input sequence, whereas $\delta(i)$ and $\eta(i, k)$ are binary variables. Vertices corresponding to μ variables are therefore

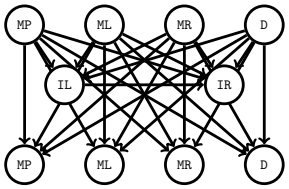
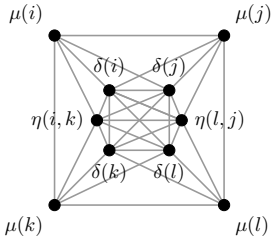
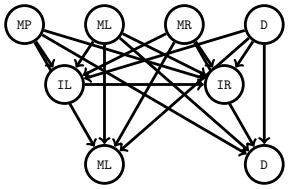
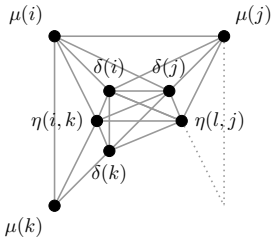
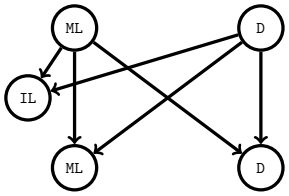
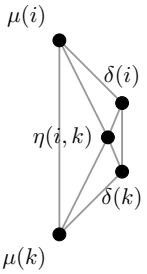
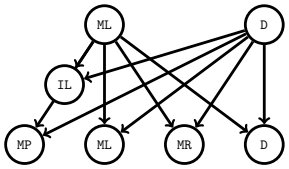
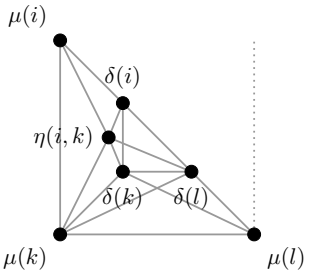
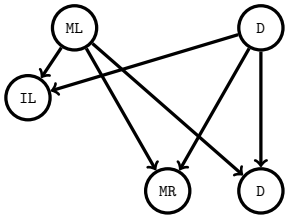
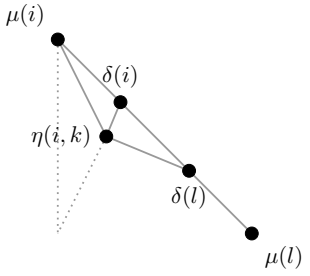
transition	state graph	variable graph
MATP→MATP		
MATP→MATL		
MATL→MATL		
MATL→MATP		
MATL→MATR		

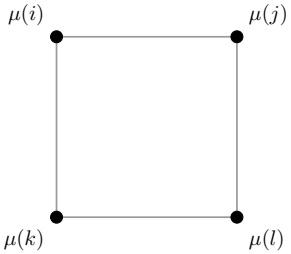
Table 4.3: Within an helix, the different possible transitions between MATP, MATL and MATR nodes give rise to different terms in the overall scoring function. A crucial feature is accounting for stacking effects in the MATP → MATP transition. In the right column, variables that appear together in a scoring term are connected by an edge. A dotted edge indicates a connection to the next μ variable. These graphs are valid for both InfeRNAl and pseudoknotted covariance models, as the scoring function is the same (the difference being the potential conflicts of helices).

said to be *heavy*, and vertices corresponding to δ and η variables are said to be *light*. Given a tree decomposition $\mathcal{T}$ of $G^+(\mathcal{A})$, we define w_{light} (resp. w_{heavy}) to be the maximum number of light vertices (resp. heavy) in a bag of $\mathcal{T}$, minus 1. The complexity of aligning a sequence S onto a PKCM depends on the following graph-theoretical result:

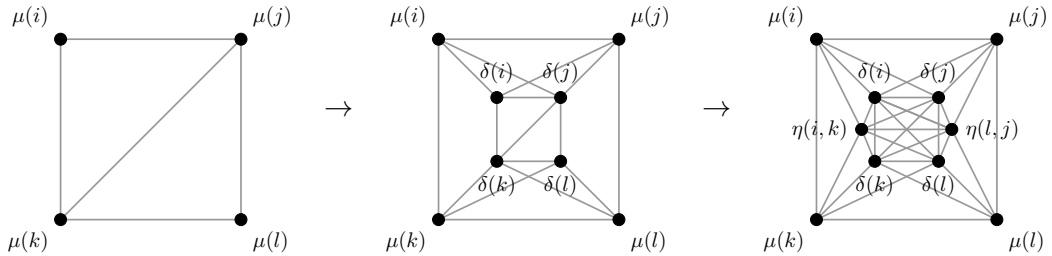
Theorem 18. *There is a constant c such that, for any $\mathcal{A}$ a pseudoknotted secondary structure and $G(\mathcal{A})$ its structure graph, there is a tree decomposition $\mathcal{T}$ of $G^+(\mathcal{A})$ such that:*

- $w_{\text{heavy}}(\mathcal{T}) = tw(G(\mathcal{A}))$
- $w_{\text{light}}(\mathcal{T}) \leq c \cdot tw(G(\mathcal{A}))$

Proof. Let us start with a tree decomposition $\mathcal{T}$ of $G(\mathcal{A})$, and augment $\mathcal{T}$ with light vertices until it represents $G^+(\mathcal{A})$. Throughout this proof, we simply denote $tw(G(\mathcal{A}))$ by tw. Let us first consider the case of stacked base-pairs (i, j) and (k, l). For each such stack, the edges originally represented by $\mathcal{T}$ are:



However, given the possibility of alternatively representing tree decompositions as *chordal completions* [168], i.e. the addition of edges so that every cycle of length ≥ 4 has a *chord*, we know that $\mathcal{T}$ must represent one of the edges $(\mu(k), \mu(j))$ or $(\mu(i), \mu(l))$. Let us assume w.l.o.g it is $(\mu(k), \mu(j))$. For "free" we now have the graph denoted by (1) below:



Let us start adding light vertices. Our purpose is to first start representing edges of $G^+(\mathcal{A})$ for *all stacks* at once, while keeping w_{light} under control, and w_{heavy} unchanged. $\forall i$, and for every occurrence of $\mu(i)$ in a bag, we start by adding $\delta(i)$ to that bag. The overall cost is that w_{light} go from 0 to tw, while w_{heavy} stays constant at tw. For each stack, $\mathcal{T}$ now represents the graph depicted above as (2).

Next, for all consecutive positions i, k, we add $\eta(i,k)$ to each bag where both $\mu(i)$ and $\mu(k)$ are present. Since there can be at most $\frac{tw+1}{2}$ pairs $(\mu(i), \mu(k))$ present in a given bag, the cost is an augmentation of w_{light} from tw to $\frac{3tw+1}{2}$. We get the graph the graph depicted above as (3). The only thing missing is to turn the six light vertices in the center a clique. We apply a similar transformation for each of the missing edges. Consider for example $(\delta(i), \delta(l))$. If there is no bag containing both, then there must be a *bag intersection* $S = X \cap Y$ of two adjacent bags X and Y such that S is a $(\delta(i), \delta(l))$ separator (Proposition 2, page 36 in Chapter 1). S must contain $\delta(k)$ and $\delta(j)$. Since $|S| \leq tw$, there can be at most $tw/2$ stacks for which S separates $\delta(i), \delta(l)$. Therefore, by augmenting all bags (including X and Y) between the occurrences of $\delta(i)$ and those of $\delta(l)$ with (for instance) $\delta(i)$, and doing so for each stack only increases w_{light} by a factor of 1.5. Doing so for edges $(\delta(i), \delta(l))$, $(\delta(i), \eta(l,j))$, $(\delta(k), \eta(l,j))$, $(\delta(j), \eta(i,k))$, $(\delta(l), \eta(i,k))$ and $(\eta(i,k), \eta(l,j))$ increases therefore w_{light} by a factor of $\left(\frac{3}{2}\right)^6$.

After this process, all necessary edges for all stacked base-pairs are represented in $\mathcal{T}$. We still have $w_{\text{heavy}} = tw$, while $w_{\text{light}} \leq \left(\frac{3}{2}\right)^6 \cdot \frac{3tw+1}{2}$. We apply similar transformations to represent all edges in $G^+(\mathcal{A})$, i.e. also covering the other cases of Table 4.3. Each new potentially incurs a constant-factor increase to w_{light}, while w_{heavy} stays constant. □

Complexity statement. Through dynamic programming over a tree decomposition of $G^+(\mathcal{A})$, we argue that we can align an input sequence S with a PKCM (i.e., finding optimal μ, δ, η for $\text{score}_{\texttt{pkcm}}$) in the following complexity:

Theorem 19. *Given a tree decomposition $\mathcal{T}$ of $G^+(\mathcal{A})$, the complexity of aligning a sequence S of length m onto a pseudoknotted covariance model with n nodes is $O(2^{w_{light}(\mathcal{T})} \cdot n \cdot m^{w_{heavy}(\mathcal{T})+1})$*

Proof. Much like the `LiCoRNA` algorithm, a table $c[X, (\mu, \eta, \delta)]$ is filled. X is a bag of $\mathcal{T}$ with parent P and children $Y_1, \ldots, Y_\ell$, and (μ, η, δ) are partial assignments to the vertices in $P \cap X$. With $(\mu^+, \eta^+, \delta^+)$ assignments over the full set X (which can be assumed to have only at most one extra element with respect to P), we have a recursive equation of the form:

$$c[X, (\mu, \eta, \delta)] = \max_{(\mu^+, \eta^+, \delta^+) \in \text{comp}(\mu, \eta, \delta)} \left[\sum_{i=1}^{\ell} c[Y_i, (\mu^+, \eta^+, \delta^+) \cap Y_i] + \texttt{lcost}((\mu^+, \eta^+, \delta^+), S) \right]$$

With:

- comp is a set of compatible assignments for $\{u\} = P \setminus X$, given the constraint μ, η, δ.
- `lcost` evaluates any term of the cost function for which X is the highest bag in $\mathcal{T}$ containing all variables involve in the term. Such a bag must exist, as the variables in question form a clique in $G^+(\mathcal{A})$.

The complexity analysis is similar to that of `LiCoRNA`. The minimization is over at most $4 \cdot m$ values, while there are $\leq 2^{\overset{\text{light}}{w}} \cdot m^{\overset{\text{heavy}}{w}}$ entries in the table. □

Combining Theorems 18 and 19 yields the following Corollary. In its statement, m is the length of a given sequence and n the number of in the pseudoknotted covariance model.

Corollary 2. *There is a constant c such that, for any sequence S and any pseudoknotted covariance model $\mathcal{P}$ with consensus structure $\mathcal{A}$, the complexity of aligning S on $\mathcal{P}$ is $O(2^{c \cdot tw(G(\mathcal{A}))} \cdot n \cdot m^{tw(G(\mathcal{A}))+1})$*

In particular, the degree of the polynomial does not increase with respect to `LiCoRNA`. For a conflict-free secondary structure, by choosing a simple tree decomposition that follows the guide tree, we do recover `InfeRNAl` and its $O(n^3)$ complexity[3].

4.4 Conclusion and perspectives

In this chapter, we discussed methods for pseudoknotted structure-sequence alignment, with the long-term purpose of enabling efficient pseudoknotted *homolog search*. Reaching that goal should allow for a better support of pseudoknotted families in `Rfam`, the reference database for ncRNAs. The need for *manual curation* of these families could then be lifted, and the ability to annotate a wider variety of pseudoknotted ncRNAs increased.

However, our numerical evaluation of `LiCoRNA`, a treewidth-based tool for pseudoknotted Structure-Sequence Alignment, on `Rfam` data shows the difficulty of outlining the added value of taking pseudoknots into account. Part of this difficulty might come from taking seed alignments of `Rfam` as a reference. Another difficulty in our experiment is the prohibitive computational cost of `LiCoRNA`, which required computing time on a cluster to be evaluated, when all other tools could be run on a laptop. A fuller set of experiments should allow for the tweaking of seed alignments, and find ways to accelerate pseudoknotted Structure-Sequence Alignment. For the former, an idea would be to use a *pseudoknotted* equivalent of `InfeRNAl`, which could then replace it

[3] conflict-free secondary structures have treewidth 2.

in any part of the `Rfam` pipeline for pseudoknotted families.

Formulating such a pseudoknotted covariance model, based on tree decompositions, and generalizing `InfeRNAl` with a *multiple context-free grammar* was carried out in the second part of this chapter. A natural next step is therefore the implementation, optimization (such as a refinement of the c constant in Theorem 18) and evaluation of pseudoknotted covariance models.

We conclude this chapter with a note regarding a strategy for accelerating the dynamic programming procedure of Theorem 19 with Tree-Diet (Chapter 2). In the proofs of concept of Chapter 2, the reduced-width models were applied to a *hierarchical filtering* of a sequence data-base. We explain here briefly how they could also accelerate *individual instances* of alignment.

Applying tree-diet to Theorem 19. Consider $\mathcal{A}$ a pseudoknotted secondary structure, and $\mathcal{A}'$ a reduced model with smaller treewidth ($\mathcal{A}' \subset \mathcal{A}$). Let $c_{\mathcal{A}'}[X, (\mu, \eta, \delta)]$ be an entry of the dynamic programming table of Theorem 19, with $\mathcal{A}'$ as consensus structure. Let also OPT be the maximum possible value of $c_{\mathcal{A}'}[X, (\mu, \eta, \delta)]$ when (μ, η, δ) are variable. If $\mathcal{R} = \mathcal{A} \setminus \mathcal{A}'$, then let us note $C_{\mathcal{R}}^{\max}$ the maximum possible contribution of the base-pairs in $\mathcal{R}$ to the score function. If

$$c_{\mathcal{A}'}[X, (\mu, \eta, \delta)] < \text{OPT} - C_{\mathcal{R}}^{\max}$$

then there is no need in computing the same entry in $c_{\mathcal{A}}$, as we know it will not be the optimal assignment. Bounds on the variables may for instance be inferred this way, hopefully reducing the practical complexity of the overall alignment problem.

Chapter 5

Independent set reconfiguration and RNA kinetics

This chapter is based on:

Laurent Bulteau, Bertrand Marchand, and Yann Ponty. A new parametrization for independent set reconfiguration and applications to RNA kinetics. In *IPEC 2021-International Symposium on Parameterized and Exact Computation*, 2021

links to:

↪ **conference version (IPEC '21)** https://drops.dagstuhl.de/opus/volltexte/2021/15394/pdf/LIPIcs-IPEC-2021-11.pdf

↪ **journal version (pre-print)** https://hal.science/hal-04094405

Abstract

In this paper, we study the Independent Set (IS) reconfiguration problem in graphs, and its applications to RNA kinetics. An IS reconfiguration is a scenario transforming an IS L into another IS R, inserting/removing one vertex at a time while keeping the cardinalities of intermediate sets as large as possible. We focus on the *bipartite* variant where only start and end vertices are allowed in intermediate ISs. Our motivation is an application to the *RNA energy barrier*, a classic hard problem from bioinformatics, which asks, given two RNA structures given as input, whether there exists a reconfiguration pathway connecting them and staying below an energy threshold. A natural parameter for this problem would be the difference between the initial IS size and the threshold (*barrier*).

We first show the para-NP hardness of the problem with respect to this parameter. We then investigate two new parameters, the *cardinality range* ρ and a measure of *arboricity* Φ. ρ denotes the maximum allowed size difference between an IS along the reconfiguration and a maximum IS, while

Φ is a measure of the amount of "branching" in the two input RNA structures. We show that bipartite IS reconfiguration is XP for ρ in the general case, and XP for Φ in the sub-case of bipartite graphs stemming from RNA instances.

We give two different routes yielding XP algorithms for ρ: The first is a direct $O(n^2)$-space, $O(n^{2\rho+2.5})$-time algorithm based on a separation lemma; The second builds on a parameterized equivalence with the directed pathwidth problem, leading to a $O(n^{\rho+1})$-space, $O(n^{\rho+2})$-time algorithm for the reconfiguration problem through an adaptation of a prior result by Tamaki [189]. This equivalence is an interesting result in its own right, connecting a reconfiguration problem (which is essentially a *connectivity* problem within a *reconfiguration network*) with a *structural* parameter for an auxiliary graph. For Φ, our $O(n^{\Phi+1})$-algorithm stems from seeing the problem as an instance of *minimum cumulative-cost scheduling*, and relies on a *merging* procedure that might be of independent interest. These results improve upon a partial $O(n^{2\rho+2.5})$-algorithm that only applied to the RNA case. We also demonstrate their practicality of these algorithms through a benchmark on small random RNA instances.

Contents

5.1 Introduction

Reconfiguration problems. Reconfiguration problems informally ask whether there exists, between two *configurations* of a system, a *reconfiguration pathway* entirely composed of *legal* intermediate configurations, connected by legal *moves*. In a thoroughly studied sub-category of these problems, configurations correspond to *feasible solutions* of some *optimization problem*, and a feasible solution is legal when its quality is higher than a specified threshold.

Examples and complexity. Examples of optimization problems for which reconfiguration versions have been studied include Dominating Set, Vertex Cover, Shortest Path or Independent Set, which is our focus in this article. As for *legal moves*, for problems with *vertex subsets* as solutions (informally seen as "tokens" placed on the graph), typical choices are *token sliding* (TS), *token jumping* (TJ) and *token addition removal* (TAR). In these models, tokens may only respectively slide on edges (TS), be transferred to another position (TJ) or be removed/added one at a time (TAR).

Depending on the choice of problem and reconfiguration model (e.g. TS, TJ or TAR),complexities range from polynomial (see [190] for examples) to NP-complete (for bipartite independent set reconfiguration in the TAR model [191]), and even PSPACE-complete for many of them [191, 192, 193]. Such computational hardness motivates the study of these problems under the lens of *parametrized complexity* [194, 195, 196, 192], in the hope of identifying tractable sub-regimes. Typical parameters considered by these studies focus on the value of the *quality threshold* (typically a *solution size* bound) defining legal configurations and the length of the reconfiguration sequences.

Directed pathwidth. *Directed pathwidth*, originally defined in [197] and attributed to Robertson, Seymour and Thomas, represents a natural extension of the notions of pathwidth and path decompositions to directed graphs. Like its undirected restriction, it may alternatively be defined in terms of *graph searching* [198], *path decompositions* [199, 200] or *vertex separation number* [201, 189]. An intuitive formulation can be stated as the search for a visit order of the directed graph, using as few active vertices as possible at each step, and such that no vertex may be deactivated until all its in-neighbors have been activated. Although an FPT algorithm is known for the undirected pathwidth [202], it remains open whether computing the directed pathwidth admits a FPT algorithm. XP algorithms [189, 201] are known, and have been implemented in practice [203, 204].

RNA energy barrier. As explained in Chapter 1, RNAs are single-stranded biomolecules which fold onto themselves into 2D and 3D structures through the pairing of nucleotides along their sequence [6]. Thermodynamics then favors low-energy structures, and the RNA energy barrier problem (Problem 4, Chapter 1 page 27) asks, given two structures, whether there exists a re-folding pathway connecting them that does not go through unlikely high-energy intermediate states [63, 15].

Kinetics as reconfiguration and scheduling. Interestingly, the problem falls under the wide umbrella of reconfiguration problems described above, namely the reconfiguration of solutions of optimization problems (here, energy minimization). An important specificity of the problem is that the

probability of a refolding pathway depends on the energy difference between intermediate states and the *starting point* rather than the absolute energy value. Another aspect is that since some pairings of the initial structure may impede the formation of new pairings for the target structure, it induces a notion of *precedence constraints*, and may therefore also be treated as a *scheduling* problem, as carried out in [205, 206].

Problem statement. In our work, we focus on independent set reconfigurations under the *token addition removal* (TAR) model, with only vertices from the start or end ISs (L and R) are allowed within intermediate ISs. This amounts to considering the induced subgraph $G[L \cup R]$, bipartite by construction. We write $\alpha(G)$ for the size of a maximum independent set of G (recall that $\alpha(G)$ can be computed in polynomial time on bipartite graphs).

Problem 11 (Bipartite Independent Set Reconfiguration (BISR))**.**
Input: Bipartite graph $G = (V, E)$ with partition $V = L \cup R$; integer ρ
Parameter: ρ
Output: True if there exists a sequence $I_0 \cdots I_\ell$ of independent sets of G such that

- $I_0 = L$ and $I_\ell = R$;
- $|I_i| \geq \alpha(G) - \rho, \forall i \in [0, \ell]$;
- $|I_i \triangle I_{i+1}| = 1, \forall i \in [0, \ell - 1]$.

False otherwise.

The vertices of L and R will typically respectively be called "start" and "end" vertices. We further assume that when a bipartite graph G is given, a specification of which side is L (the "start" set) and which is R (the "end" set) is given. When we explicitly need to state which sets are the "start" and "end" sets, we will write $G_{L \to R}$.

"range" definition. Figure 5.1 shows an example of an instance of BISR and a possible reconfiguration pathway. We introduce the *cardinality range* (or simply *range*) $\rho = \max_{1 \leq i \leq \ell} \alpha(G) - |I_i|$ as a natural parameter for this problem, since it measures a distance to optimality. As mentioned above, another natural parameter for RNA kinetics is the *barrier*, denoted k, and defined as $k = \max_{1 \leq i \leq \ell} |L| - |I_i|$. Intuitively, k measures the size difference from the starting point rather than from an "absolute" optimum. Note that $k = \rho - (\alpha(G) - |L|)$, so one has $0 \leq k \leq \rho$. Both parameters are obviously similar for instances where L is close to being a maximum independent set, which is generally the case in RNA applications, but in theory the range ρ can be arbitrarily larger than the barrier k.

Our results. We first prove that in general, the barrier k may not yield any interesting parameterized algorithm, since BISR is Para-NP-hard for this parameter. We thus focus on two other parame-

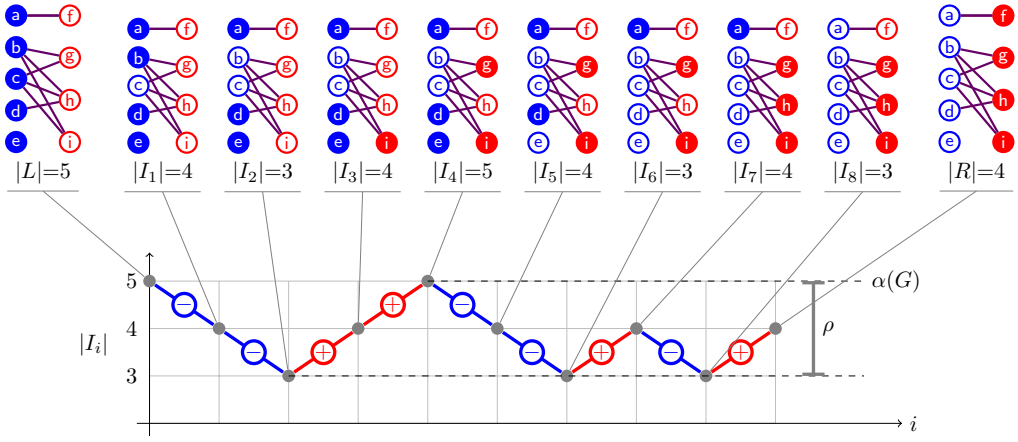


Figure 5.1: Example of a bipartite independent set reconfiguration from L (blue) to R (red). Selected vertices at each step have a filled background. All intermediate ISs have size at least 3, and the optimal IS has size 5, so this scenario has a range of 2; it can easily be verified that it is optimal.

terizations, the *range* ρ, as defined above and illustrated in Figure 5.1, and the *arboricity* Φ in the case of RNA instances, as illustrated on Figure 5.7.

Regarding ρ. For the range ρ, we prove that BISR is in XP by providing two distinct algorithmic strategies to tackle it. Our first algorithmic strategy stems from a parameterized equivalence we draw between BISR parameterized by ρ and the problem of computing the directed pathwidth of directed graphs. It is obtained by defining a directed graph from a *maximum matching* of the input bipartite graph. Within this equivalence, ρ maps exactly to the directed pathwidth. This allows to apply XP algorithms for Directed Pathwidth to BISR while retaining their complexity, such as the $O(n^{\rho+2})$-time, $O(n^{\rho+1})$-space algorithm from Tamaki [189] (with $n = |V|$). This equivalence between directed pathwidth and bipartite independent set reconfiguration is itself an interesting result, as it connects a *structural* problem, whose parameterized complexity is open, with a reconfiguration problem of the kind that is routinely studied in parameterized complexity [194, 195, 196, 192].

The other algorithmic strategy for BISR parameterized by ρ is more direct, and runs with a time complexity of $O(n^{2\rho}\sqrt{n}m)$ ($m = |E|$) but using only $O(n^2)$ space. It is an example of the "bounded search-tree" technique [71], enabled by a *separation lemma*. This lemma involves, if it exists, a *mixed* maximum independent set of G containing at least one vertex from both parts of the graph. In the specific case of bipartite graphs arising from RNA reconfiguration, we improve the run-time of the subroutine computing a mixed MIS to $O(n^2)$ (rather than $O(\sqrt{n}m)$), with a dynamic programming approach.

Regarding Φ. As for the arboricity Φ, we also show membership in XP of BISR restricted to RNA instances, through dynamic programming over the sub-trees of $T(S)$ for one of the input structures S (see Figure 5.7). This XP algorithm we present involves seeing the problem in terms of *cumulative cost-*

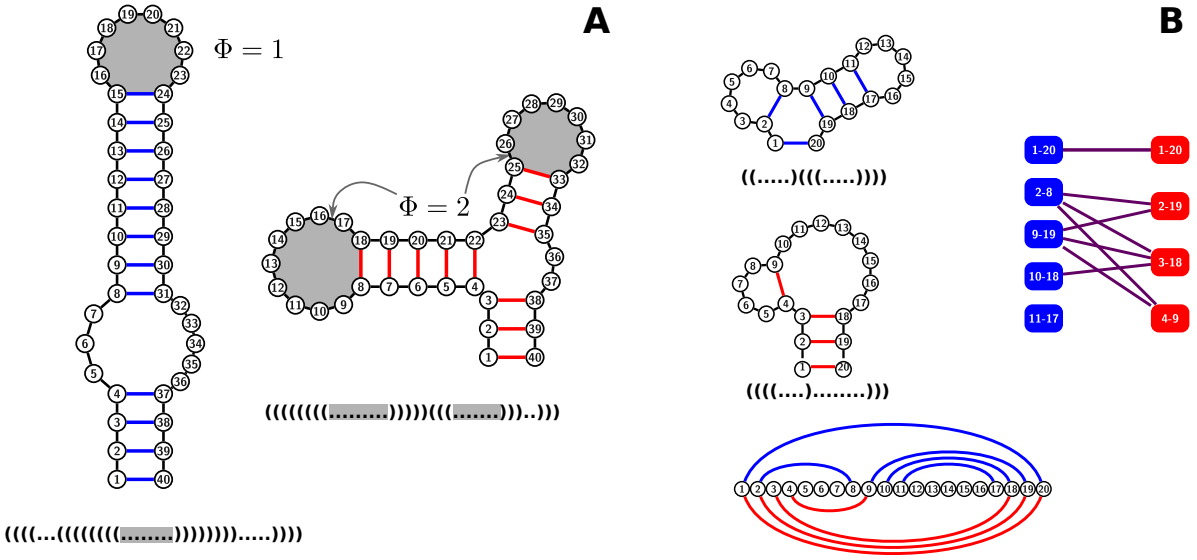


Figure 5.2: (A) Example of two RNA structures, and the corresponding value for the arboricity Φ. Within this work, we consider only "conflict-free" secondary structures that can be seen as well-parenthesized strings. Φ is then the number of "terminal" pairs of matching parentheses, highlighted by gray shading. (B) Example of a conflict bipartite graph associated to two input secondary structures. The arcs (base-pairs) of both structures are the vertices of the graph, and two vertices are in conflict if the corresponding arcs are not nested in one another. By "RNA instances of BISR", we mean instances of BISR in which the bipartite graph is such a conflict graph of two RNA structures.

optimal scheduling [207]. In particular, we develop a *merge* procedure for combining optimal solutions on disjoint graphs into an optimal solution for the union of the graphs. We believe this merge procedure and its associated concepts (canonical solutions, preferability criteria) can be of independent interest.

Benchmark results. Finally, we present benchmark results for all algorithms, on random instances of general bipartite graphs as well as instances of the RNA Energy Barrier problem. The approach based on directed pathwidth yields reasonable solving times for RNA strings of length up to ~ 150 nucleotides.

Outline of this chapter. To start with, Section 5.2 presents some previously known results related to BISR, and some notations and definitions we will use throughout the article. Then, Section 5.3 shows that BISR is equivalent to the computation of *directed pathwidth* in directed graphs. Section 5.4 presents the separation lemma and merge procedure on which our direct XP algorithm in ρ and our XP algorithm in Φ are based. The related concepts of canonical schedule and preferability between schedules are also introduced in this section. Section 5.5 and Section 5.6 build on the technical results of Section 5.4 to present our direct XP algorithm parameterized by ρ and our algorithm XP in Φ in the RNA case. To finish, Section 5.7 explains some optimizations specific to RNA reconfiguration instances, and presents our numerical results.

5.2 Preliminaries

5.2.1 State of the art

Computational hardness. Bipartite Independent Set Reconfiguration was proven NP-complete in [191], through the equivalent k-Vertex Cover Reconfiguration problem. Formulated in terms of RNAs (RNA Energy Barrier), and restricted to secondary structures (i.e., the subset of bipartite graphs that can be obtained in RNA reconfiguration instances), it was independently proven NP-hard in [63]. To the authors' knowledge, its parameterized complexity remains open.

General independent set reconfiguration. Independent set reconfiguration in an unrestricted setting (allowing vertices which are outside from the start or end independent sets, i.e. in possibly non-bipartite graphs) when parameterized by the minimum allowed size of intermediate sets has been proven W[1]-hard [194, 192], and fixed-parameter tractable for planar graphs or graphs of bounded degree [195]. Whether this more general problem is in XP for this parameter remains open. We note that in this setting, parameter ρ seems slightly less relevant since it involves computing a maximal independent set in a general graph (i.e. testing if there exists a reconfiguration from $\emptyset$ to $\emptyset$ with range ρ is equivalent to deciding if $\alpha(G) \geq \rho$).

Heuristics. Given the great practical importance of RNA Energy Barrier (See Chapter 1, Section 1.1.4.4) in Bioinformatics (BISR in the RNA case), several heuristics [65, 208] have been developed for it. In this paper, we assess the potential of parameterized algorithmics for the development of efficient *exact* algorithms for the problem, starting with a simple energy model (corresponding to Bipartite Independent Set Reconfiguration).

Exact algorithms. As for exact algorithms for BISR, the closest precedent is an algorithm by Thachuk et al. [15]. It is restricted to RNA secondary structure conflict graphs, and additionally to conflict graphs for which both parts L and R are *maximum independent sets* of G. In this restricted setting, although it is not stated as such, [15] provides an XP algorithm with respect to the barrier parameter k which then coincides with the range parameter ρ that we introduce. In this paper, we extend this line of study by showing the Para-NP-hardness of BISR for k in the general setting. We further show that generalizing k into ρ allows to retain membership in XP.

Bounded Φ case. Recent unpublished work [206, 205] describe polynomial-time algorithm for restricted input versions of BISR. More precisely, within RNA instances of BISR, they tackle the $\Phi = 1$ case, called "bipartite permutation graphs" in [206] and "convex bipartite partial orders" in [205]. The specialization of XP algorithm we describe for Φ yields a $O(n^3)$ algorithm in that case, improving over the $O(n^6 \log n)$ of [206].

5.2.2 Preliminary results and notations

Restriction to the monotonous case. A reconfiguration pathway for bipartite independent set reconfiguration is called *monotonous* or *direct* if every vertex is added or removed exactly once in the entire sequence. The length of a monotonous sequence is therefore necessarily: $\ell = |L \cup R| = |L| + |R|$. Theorem 2 from [191] tells us that if G, ρ is a yes-instance of bipartite independent set reconfiguration, then there exists a *monotonous* reconfiguration between L and R respecting the constraints. We will therefore restrict without loss of generality our study to this simpler case. In the more restricted set studied in [15], this was also independently shown.

Hardness for the barrier parameter. In the general case where L is not necessarily a maximal independent set, the range and barrier parameters (respectively ρ and $k = \rho - (\alpha(G) - |L|)$ may be arbitrarily different. The following result motivates our use of parameter ρ for the parameterized analysis of BISR.

Proposition 8. BISR *is* Para-NP*-hard for the energy barrier parameter k (i.e.,* NP*-hard even for a constant value of k, here with $k = 0$).*

Proof. We use additional vertices in R to prove this result. Informally, such a vertex may always be inserted first in a realization: it improves the starting IS from $|L|$ to $|L| + 1$, so the lower bound on the

rest of the sequence is shifted from $|L| - k$ to $|L| - (k-1)$, effectively reducing the barrier without simplifying the instance. Thus, we build a reduction from the general version of BISR: given a bipartite graph G with parts L and R and an integer ρ, we construct a new instance G' with parts $L' = L$ and R' equal to $R \cup N_R$ and $\rho' = \rho$. N_R is composed of $|L| - (\alpha(G) - \rho)$ isolated vertices (we can assume without loss of generality that this quantity is non-negative, otherwise (G, ρ) is a trivial no-instance), completely disconnected from the rest of the graph.

Note that $\alpha(G') = \alpha(G) + |N_R| = |L| + \rho$, so the barrier in (G', ρ') is $k = \rho - (\alpha(G') - |L|) = 0$. A realization for (G, ρ) can be transformed into a realization for (G', ρ) by inserting vertices from N_R first, and conversely, vertices from N_R can be ignored in a realization for (G', ρ) to obtain a realization for (G, ρ). Therefore, since BISR is NP-Complete, it is also Para-NP-hard w.r.t the barrier k. □

5.2.3 Definitions

Licit sets and permutations. The following definitions and notations will be used throughout the chapter. They allow to link, at an intermediary step along a reconfiguration, the set of processed vertices to the current independent set and its size. We first start with the central definition of *licit subset* and *licit permutation*. It uses the following notation: given a subset X of vertices, we define $I(X) = (L \setminus X) \cup (R \cap X) = L \Delta X$. Intuitively, in a bipartite graph G with sides L and R, $I(X)$ is the independent set obtained after processing the vertices of X, starting from L ($L \cap X$ removed, $R \cap X$ added).

Definition 26 (licit subset)**.** Given a bipartite graph G with sides L and R, and $X \subseteq L \cup R$ We say that X is *licit* if $I(X)$ is an independent set.

Permutation formulation. An equivalent representation of a monotonous reconfiguration pathway $I_0 \dots I_\ell$ from L to R for a graph G is a *permutation* S of $L \cup R$. We will also use the term of *schedule*. The i-th vertex of the permutation is the vertex that is *processed* (i.e. added or removed) between I_{i-1} and I_i (this formulation lightens the representation of a solution, from a list of vertex sets to a list of vertices). We write $P \sqsubseteq S$ if P is a prefix of S, and $V(P)$ (or simply P if the context is clear) for the set of vertices appearing in P. The search space of BISR is made of *licit permutations*, which we formally define below:

Definition 27 (licit permutation)**.** A permutation S is licit if $V(P)$ is licit for each prefix P of S

Note that S is licit if and only if $\forall r \in R$, the neighborhood $N(r)$ of r in G appears before r in S.

Balance δ and ρ-realizations. Given a subset X of vertices, we write $\delta(X) = |L \cap X| - |R \cap X|$. With this quantity, $|I(X)| = |L| - \delta(X)$. $\delta(X)$ is called the *balance* of X, as it corresponds to the size difference between the initial IS L and the current IS $I(X)$. Then, S is a permutation (or *schedule*) of

barrier k if S is licit and for each prefix $P \sqsubseteq S$, $\delta(V(S)) \leq k$. Equivalently, S is a *ρ-realization* if S is licit and such that for each prefix $P \subseteq S$, $|I(P)| \geq \alpha(G) - \rho$ (i.e. $\delta(V(P)) \leq \rho + |L| - \alpha(G)$). This is consistent with the fact that $\rho = k + \alpha(G) - |L|$.

Budget. Finally, given a bipartite graph G and a licit permutation S for G, we write $bg(S)$ for the barrier of S, i.e.

$$bg(S) = \max_{P \sqsubseteq S} |I(P)| - |L| = \max_{P \sqsubseteq S} \delta(P)$$

The *budget* of a graph G is the best possible budget of a licit permutation of G. Denoting by $\mathcal{L}(G)$ the set of licit permutations of G:

$$bg(G) = \min_{S \in \mathcal{L}(G)} bg(S)$$

A related quantity is the best possible *range* of a graph G, which we write $\rho(G)$. It verifies $\rho(G) = bg(G) + \alpha(G) - |L|$. Note that with these definitions, the BISR problem can be equivalently defined as deciding, given a graph G and an upper-bound ρ, whether $\rho(G) \leq \rho$.

5.3 Connection with Directed Pathwidth

We first present a parameterized reduction from bipartite independent set reconfiguration to an input-restricted version, on bipartite graphs allowing for a perfect matching. Then, this version of the problem is shown to be simply equivalent to the computation of directed pathwidth on general directed graphs.

5.3.1 Definitions

Parameterized reduction. In this section, we provide a definition of directed pathwidth, and then prove its parameterized equivalence to the bipartite independent set reconfiguration problem. We say two problems $\mathcal{P}_1$ and $\mathcal{P}_2$ are parametrically equivalent when there exists both a *parameterized reduction* from $\mathcal{P}_1$ to $\mathcal{P}_2$ and another from $\mathcal{P}_2$ to $\mathcal{P}_1$. The notion of *parameterized reduction* is also touched upon in Chapter 1, Section 1.2.1. In this chapter, we simply use the fact that a sufficient condition to obtain a parameterized reduction [71] from problem $\mathcal{P}$ to problem $\mathcal{Q}$ is to have a function φ from instances of $\mathcal{P}$ to instances of $\mathcal{Q}$ such that (i) $\varphi(x)$ is a yes-instance of $\mathcal{Q} \Leftrightarrow x$ is a yes-instance of $\mathcal{P}$, (ii) φ can be computed in polynomial time (iii) the parameter of x and the parameter of $\varphi(x)$ are equal.

Interval representation. Our definition of directed pathwidth relies on interval embeddings. Alternative definitions can be found, for instance in terms of directed path decomposition or directed vertex separation number [198, 189, 201].

Definition 28 (Interval representation)**.** An *interval representation* of a directed graph H associates each vertex $u \in H$ with an interval $I_u = [a_u, b_u]$, with a_u, b_u integers. An interval representation is *valid* when $(u, v) \in E \Rightarrow a_u \leq b_v$. I.e, the interval of u must start before the interval of v ends. If m, M are such that $\forall u, m \leq a_u, b_u \leq M$, we define the *width* of an interval representation as $\max_{m \leq i \leq M} |\{u | i \in I_u\}| - 1$

Definition 29 (directed pathwidth)**.** The *directed pathwidth* of a directed graph H is the minimum possible width of a valid interval representation of H. We note this number $dpw(H)$.

A proof of the equivalence of this definition with other formulations of directed pathwidth is given in the Appendix (Proposition 14).

Nice interval representation. An interval representation is said to be *nice* when no more than one interval bound is associated to any given integer, and the integers associated to interval bounds are exactly $[1 \dots 2 \cdot |V(H)|]$. Any interval representation may be turned into a nice one without changing the width by introducing new positions and "spreading events". See Appendix D.4 for more details.

Directed graph from perfect matching. Given a bipartite graph G allowing for a perfect matching M, we construct an associated directed graph H in the following way: the vertices of H are the edges of the matching, and $(l, r) \to (l', r')$ is an arc of H iff $(l, r') \in G$. Alternatively, H is obtained from G, M by orienting the edges of G from L to R, and then contracting the edges of M. We will denote this graph $H(G, M)$, and simply call it the *directed graph associated to* G, M. Such a construction is relatively standard and can be found in [209, 210], for instance.

5.3.2 Directed pathwidth ⇔ Bipartite independent set reconfiguration

Perfect matching case. Our main structural result regarding directed pathwidth is the following. Its proof relies on interval representations, with the intuition that the number of open intervals at a given position is the number of dependencies that have been lifted, but not compensated for yet.

Proposition 9. *Let G be a bipartite graph allowing for a perfect matching M. Then G allows for a ρ-realization iff $dpw(H(G, M)) \leq \rho$.*
Conversely, given any directed graph H, there exists a bipartite graph G allowing for a perfect matching M such that $H = H(G, M)$ and G allows for a ρ-realization iff $dpw(H) \leq \rho$.

Proof. We start with the first statement, the equivalence between $dpw(H(G, M)) \leq \rho$ and the existence of a ρ-realization for G. First note that, since G allows for a perfect matching, we have $|L| = |R|$, and by König's theorem, if K is a minimum vertex cover of G, $|K| = |L| = |R|$. Since

$\alpha(G) = |L| + |R| - |K|$ we have $\alpha(G) = |L| = |R|$. I.e. L and R are maximum independent sets of G.

$\Rightarrow$ If G allows for a ρ-realization, then $\exists P$ ordering of the vertices of G such that every prefix X_i of P verifies $|I(X_i)| = |L| - \delta(X_i) = \alpha(G) - \delta(X_i) \geq \alpha(G) - \rho$. Therefore $\delta(X_i) = |X_i \cap L| - |X_i \cap R| \leq \rho$.

Consider a vertex (l, r) of $H(G, M)$, with (l, r) an edge of M. We associate to (l, r) the interval $[a_{(l,r)}, b_{(l,r)}]$ where $a_{(l,r)}$ is such that $P[a_{(l,r)}] = l$. i.e, it corresponds to the step in the reconfiguration where l is removed. Likewise, $b_{(l,r)}$ is such that $P[b_{(l,r)}] = r$.

For any edge $(l, r) \to (l', r')$ of H, necessarily $(l, r') \in G$, which implies that in the reconfiguration sequence, l has to be removed before r' is added. l appears therefore earlier than l in P, and $a_{(l,r)} \leq b_{(l',r')}$. The intervals we have defined therefore form a valid interval representation of H.

In addition, the intervals intersecting a given position i correspond to pairs (l, r) where, at step i, l has already been removed while r is yet to be added.

Since the decrease in independent set size incurred by the removal of l is compensated by the addition of its match r, the number of intervals intersecting position i is exactly $\delta(X_i)$, the imbalance of the i-prefix of P, which by hypothesis is $\leq \rho$.

$\Leftarrow$ Suppose the directed graph $H(G, M)$ associated to G, M has directed pathwidth $\leq \rho$. Consider an optimal nice interval representation for H.

In this representation, a vertex (l, r) of H is associated to an interval $[a_{(l,r)}, b_{(l,r)}]$. Thanks to the structure of *nice* interval representation, we simply define a permutation P of $L \cup R$ with, $\forall (l, r)\ P[a_{(l,r)}] = l$ and $P[b_{(l,r)}] = r$.

If (l, r') is an edge of G, with r the match of l and l' the match of r', then the construction above ensures that l is before r' in P. For two matched vertices, this is also immediate. Then, as for two matched vertices l, r, the removal of l is compensated by the addition of r, for any prefix X_i of P, the imbalance $\delta(X_i)$ is exactly the number of intervals intersecting position i. By assumption, we therefore have $\delta(X_i) \leq \rho$ and P is a ρ-realization.

For the second part of the statement, given a directed graph H, we construct a bipartite graph G with sides L, R allowing for a perfect matching M in the following way: for each vertex $u \in H$ we introduce two vertices (l_u, r_u) in G. We assign l_u to L and r_u to R, connect l_u and r_u and add the edge to the matching M. We now add an edge from l_u to r_v in G for any $(u, v) \in E(H)$. G now verifies $H = H(G, M)$, and by the result above, $dpw(H) \leq \rho$ iff G allows for a ρ-realization.

□

The first half of Proposition 9 is a parameterized reduction from an input-restricted version of BIPARTITE INDEPENDENT SET RECONFIGURATION to directed pathwidth. The restriction is on bipartite graphs allowing for a perfect matching. The second half is a parameterized reduction in the other

direction. In both cases, the parameter value is directly transferred, which allows to retain the same complexity when transferring an algorithm from one problem to the other.

Non-perfect-matching case. In the case where G does not allow for a perfect matching, we construct G' allowing for a perfect matching M', and such that $\rho(G) = \rho(G') = dpw(H(G', M'))$. G' is obtained from G through the addition of new vertices. Specifically, with a bipartite graph G with sides L, R, a maximum matching M of G, and the set U of unmatched vertices in G, we extend G with $|U|$ new vertices in two sets N_L, N_R, giving a new graph G', with sides $L' = L \cup N_L, R' = R \cup N_R$, in the following way ($M'$ is initialized to M):

- For each $u \in L \cap U$, we introduce a new vertex $r(u) \in N_R$, connect it to *all* vertices of L', and add the edge $(u, r(u))$ to M'.
- Likewise, for each $v \in R \cap U$, we introduce $l(v) \in N_L$, connect it to all vertices of R' and add $(v, l(v))$ to M'.

Note that M' is a perfect matching of the extended bipartite graph G'.

Proposition 10. *With G, G' defined as above, we have that G allows for a ρ-realization iff G' allows for a ρ-realization.*

Proof. First note that by König's Theorem, $\alpha(G') = |M'| = |M| + |U| = \alpha(G)$, so it suffices to ensure that any realization for G can be transformed into a realization for G' where independent sets are lower-bounded by the same value, and vice versa.

Let P be any ρ-realization of G, then $P' = N_L \cdot P \cdot N_R$ is a ρ-realization for G', with N_L and N_R laid out in any order. Indeed, P' satisfies the precedence constraint, and any intermediate set I in P' satisfies one of the following cases: $L \subseteq I$, $R \subseteq I$, or I is an intermediate set from P, so in any case it has size at least $\alpha(G) - \rho = \alpha(G') - \rho$.

Conversely, because of the all-to-all connectivity between N_L and R and between L and N_R, a realization for G' needs to have N_L before any vertex from R, and have N_R after all vertices from L. Without loss of generality, it is therefore of the form $N_L \cdot P \cdot N_R$ with P a realization of G, and G allows for a ρ-realization. □

The construction above in fact yields a parameterized reduction from BIPARTITE INDEPENDENT SET RECONFIGURATION to its input-restricted version on bipartite graphs allowing for a perfect matching. This input-restricted version is in turn parametrically equivalent to directed pathwidth by Proposition 9. Hence the following corollary:

Corollary 3. BIPARTITE INDEPENDENT SET RECONFIGURATION *is parametrically equivalent to* DIRECTED PATHWIDTH

It allows to import algorithmic results for DIRECTED PATHWIDTH and apply them to BIPARTITE INDEPENDENT SET RECONFIGURATION. In particular:

Corollary 4. *There exists a $O(n^{\rho+1})$-space, $O(n^{\rho+2})$-time* XP *algorithm for* Bipartite Independent Set Reconfiguration.

Proof. Application of the algorithm from [211]. See also Section 5.7.1 for more details. □

An implementation and a benchmark of this algorithm in the context of RNA kinetics is presented in Section 5.7.

Limitations. The high space-complexity of [211] may hinder the practicality of the algorithm, and the ρ parameterization may not necessarily be adapted to the RNA kinetics context. This is why we also explored direct algorithms parameterized by ρ for the problem, presented in Section 5.5, and another parameterization (arboricity) in Section 5.6. They rely on the technical elements presented in the following section, regarding the role of *mixed* maximum independent sets (i.e., intersecting both L and R) in G as *separators* and the problem of optimally *merging* optimal solutions for disconnected instances.

5.4 Lemmata: algorithmic building blocks

This section introduces our main technical results, which are the building blocks of the algorithms we propose for Bipartite Independent Set Reconfiguration. They consist of a *separation lemma* and a *merge* procedure.

5.4.1 Definitions

Separators in BISR instances. We use the permutation representation of reconfiguration scenarios, i.e. licit permutations of vertices (Definition 27). It will allow us to define *separators*, i.e. splitting points in BISR instances, which will enable a divide and conquer approach.

Licit subset combination. First note that the intersection, as well as the union, of two licit set of vertices are licit:

Property 5. *Given X, Y two licit subsets of a graph G, both $X \cap Y$ and $X \cup Y$ are licit subsets.*

Proof. Let us check that $I(X \cap Y) = L \setminus (X \cap Y) \cup (R \cap X \cap Y)$ is indeed an independent set. Consider $r \in R$ and suppose $r \in I(X \cap Y)$. Then $r \in R \cap X \cap Y$, and since X is licit, $r \in X \cap R$ implies $N(r) \cap I(X) = \emptyset$ and therefore $N(r) \subset L \cap X$. Likewise, $N(r) \subset L \cap Y$. Therefore $N(r) \subset L \cap X \cap Y$ and $I(X \cap Y)$ does not contain $N(r)$. Likewise, consider $\ell \in L$, and suppose $\ell \in I(X \cap Y)$. Then $\ell \in L \setminus (X \cap Y)$, so either $\ell \in L \setminus X$ or $\ell \in L \setminus Y$. Since X and Y are licit, either $N(r) \cap (R \cap X) = \emptyset$ or $N(r) \cap (R \cap Y) = set$. In any case, $N(\ell) \cap I(X \cap Y) = \emptyset$ and $I(X \cap Y)$ is indeed an independent set.

Let us now check that $I(X \cup Y)$ is also an independent set. In a similar fashion, consider $r \in I(X \cup Y) \cap R$. $r \in R \cap (X \cup Y)$ implies $r \in R \cap X$ or $r \in R \cap Y$, which implies since X and Y are licit $N(r) \cap (L \setminus X) = \emptyset$ or $N(r) \cap (L \setminus Y) = \emptyset$, i.e. $N(r) \subseteq X$ or $N(r) \subseteq Y$. In any case, $N(r) \cap (L \setminus (X \cup Y)) = \emptyset$.

To finish, consider $\ell \in I(X \cup Y) \cap L$. We have $\ell \in L \setminus (X \cup Y)$, so $\ell \notin X$ and $\ell \notin Y$. Since X and Y are licit, $N(\ell) \cap X = \emptyset$ and $N(\ell) \cap Y = \emptyset$, so $I(X \cup Y) \cap N(\ell) = R \cap (X \cup Y) \cap N(\ell) = \emptyset$, and $I(X \cup Y)$ is indeed and independent set.

□

Permutation sub-sampling. In order to define separators, we describe here some notations useful for *re-ordering* operations in permutations. Given a realization P of G and a set of vertices X, we write $P \cap X$ for the sub-sequence of P consisting of the vertices of X, without changing the order. Likewise, $P \setminus X$ denotes the sub-sequence of P consisting of vertices *not* in X.

Notation 5. *$P \cap X$ and $P \setminus X$*

Definition of separators. A *mixed maximum independent set* I of G is an independent set of G of maximum cardinality containing at least a vertex from both parts. Note that not every bipartite graph contains such a set. Separators are licit subsets whose processing leads to mixed maximum independent sets.

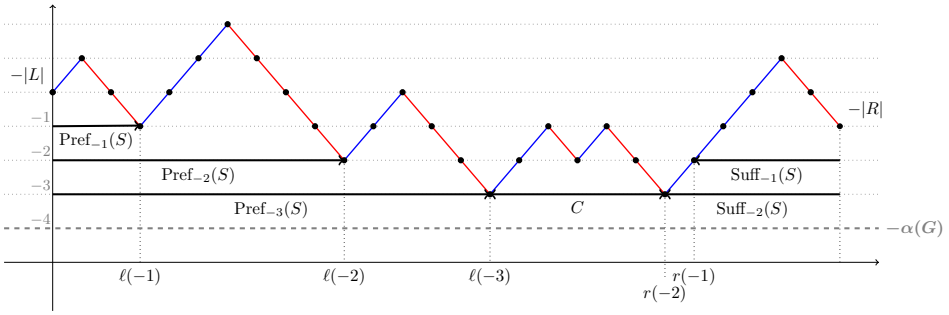
Definition 30 (separator)**.** A *separator* X is a subset of $L \cup R$ such that $I(X)$ is a mixed maximum independent set of G.

Separators and inversions. When otherwise specified, a bipartite graph G has sides L and R, and the independent set reconfiguration goes from L to R. In the proofs below however, it will be useful to also consider the instance of BISR in which R is reconfigured into L. To differentiate both, given a graph G, we write $G_{L \to R}$ (resp. $G_{R \to L}$) or the instance of BISR in which L is reconfigured into R (resp. R into L). Likewise, we write $I_{L \to R}(X) = (L \setminus X) \cup (R \cap X)$ to denote the independent set obtained by processing X starting from L, while $I_{R \to L} = (R \setminus X) \cup (L \cap X)$ is the result of processing X starting from R. Interestingly, a separator for $G_{L \to R}$ is then also a separator for $G_{R \to L}$.

Property 6. *Let X be a separator of $G_{L \to R}$. Then $Y = G \setminus X$ is a separator of $G_{R \to L}$*

Proof. $I_{R \to L}(Y) = (R \setminus Y) \cup (L \cap Y) = (R \setminus (G \setminus X) \cup (L \cap (G \setminus X))) = (R \cap X) \cup (L \setminus X) = I_{L \to R}(X)$, which is a mixed maximum independent set of $G_{L \to R}$ (same graph as $G_{R \to L}$) □

Preferability and canonical schedules. The two technical results presented in this section, Lemma 5.4.2 (separation Lemma) and Theorem 20 (merging procedure) are expressed in terms of a notion of *preferability* and *canonical schedules*. The preferability order relation allows to choose between different schedules equivalent in terms of barrier, while a canonical schedule is the "most preferable". These two notions are defined in the following paragraphs.



$-i$	$\mathrm{pref}_{-i}(S)$	$\mathrm{suff}_{-i}(S)$
-1	1	2
-2	2	2
-3	2	$+\infty$
-4	$+\infty$	undefined

Figure 5.3: Illustration of the definition of $\mathrm{pref}_{-i}(S)$ and $\mathrm{suff}_{-i}(S)$

level-specific budgets. Formally, given a schedule S for a graph G, and $-1 \geq -i \geq |L| - \alpha(G)$, we define $\ell_S(-i)$ as the smallest strictly positive integer, if it exists, such that $\delta(S_{\leq \ell_S(i)}) = -i$. Likewise, for i such that $-1 \geq -i \geq |R| - \alpha(G)$ we define $r_S(-i)$ as the largest integer $< |S|$, if it exists, such that $\delta(S_{\leq r_S(-i)}) + |R| - |L| = i$. We then write

$$\mathrm{pref}_{-i}(S) = bg(S_{\leq \ell_S(-i)})$$

and

$$\mathrm{suff}_{-i}(S) = bg(S_{\geq r_S(-i)}) - \delta(r_S(-i))$$

. The corresponding prefixes and suffixes are denoted by $\mathrm{Pref}_{-i}(S)$ and $\mathrm{Suff}_{-i}(S)$. If $\ell_S(-i)$ does not exist, then $\mathrm{pref}_{-i}(S) = +\infty$, and likewise for $r_S(-i)$ and $\mathrm{suff}_{-i}(S)$. Informally, these quantities are "the budget needed to reach level $-i$" in the forward and reverse directions. These definitions are illustrated in Figure 5.3. Note that upon inverting the start set L and final set R, then $\ell_S(-i), \mathrm{Pref}_{-i}(S)$ and $\mathrm{pref}_{-i}(S)$ become $r_S(-i), \mathrm{Suff}_{-i}(S), \mathrm{suff}_{-i}(S)$ and vice-versa. To be more precise, given S a schedule, i.e. an order on vertices, let us denote by $\overleftarrow{S}$ its reverse schedule, with opposite order. Then, we have:

$$\mathrm{Pref}_{-i}(S) = \overleftarrow{\mathrm{Suff}_{-i}(\overleftarrow{S})}$$

and

$$\mathrm{pref}_{-i}(S) = \mathrm{suff}_{-i}(\overleftarrow{S})$$

Based on these quantities, we define a partial order on schedules for G:

Definition 31 (preferability)**.** Given S and S' two schedules for a bipartite graph G, we say that S is *preferable* to S' (denoted $S \preccurlyeq S'$) if $bg(S) \leq bg(S')$ and $\forall i$, $\mathrm{pref}_{-i}(S) \leq \mathrm{pref}_{-i}(S')$ and $\mathrm{suff}_{-i}(S) \leq \mathrm{suff}_{-i}(S')$. In the case of an equality for all criteria, $S \preccurlyeq S'$ if $\forall i$, $\ell_S(-i) \leq \ell_{S'}(-i)$ and $r_S(-i) \geq r_{S'}(-i)$.

Finally, we say that S is *strictly preferable* to S' if $S \preccurlyeq S'$ and $S' \not\preccurlyeq S$

Remark that this relation is neither total nor antisymmetric (there are pairs S, S' with either both $S \preccurlyeq S'$ and $S' \preccurlyeq S'$ or both $S \not\preccurlyeq S'$ and $S' \not\preccurlyeq S'$), but it is easily seen to be transitive from the definition. We are mostly interested in the search of global optimums, as formulated below.

Definition 32 (canonical solution)**.** A schedule S that is preferable to any other schedule S' for G is called a *canonical solution* for G.

The notion of preferability gives us a simple criteria to choose between different schedules with the same overall budget. This will be exploited algorithmically in our dynamic programming approach to the bipartite independent set reconfiguration problem in the case for bipartite circle graphs, presented in Section 5.6.

5.4.2 Separation lemma

Modularity. Lemma 5.4.2 (below) on which our algorithm XP in ρ is based is proved using the following "modularity" property of the balance functions. Interestingly, it is almost the same property (sub-modularity), on a different quantity (the in-degrees of vertices) on which rely the XP algorithm for directed pathwidth [189].

Lemma 5 (modularity)**.** *Given licit subsets X and Y, we have:*

$$|I(X)| + |I(Y)| = |I(X \cup Y)| + |I(X \cap Y)|$$

and

$$\delta(X \cup Y) + \delta(X \cap Y) = \delta(X) + \delta(Y)$$

Proof. We have $I(X) = (L \setminus X) \cup (R \cap X)$. Therefore, $|I(X)| = |L \setminus X| + |R \cap X| = |L| - |L \cap X| + |R \cap X|$. Furthermore, $|(X \cup Y) \cap L| = |(X \cap L) \cup (Y \cap L)| = |X \cap L| + |Y \cap L| - |X \cap Y \cap L|$, and likewise for R. The result stems from a substraction of one equation to the other, and an addition of $|L|$. As for the second part, it comes from the definition $\delta(X) = |L| - |I(X)|$. □

Separation lemma. Based on this "modularity", the following separation lemma is shown by "re-shuffling" a solution into another one going through a mixed MIS.

Lemma 6 (separation lemma)**.** *Let X be a separator of G. If S is a schedule for G, then $(S \cap X) \cdot (S \setminus X) \preccurlyeq S$.*

Proof. Let us write $S' = (S \cap X) \cdot (S \setminus X)$, and start by showing $bg(S') \leq bg(S)$. Let $\rho' \sqsubseteq S'$. We first introduce the following notation: given a prefix ρ' of S' with $(S \cap X) \sqsubseteq \rho'$, we write $\mathrm{rem}_X(\rho')$ for the smallest prefix ρ of S such that $V(\rho \cup X) = V(\rho')$. This definition is illustrated on Figure 5.4.

- if $\rho' \sqsubseteq (S \cap X)$, then $\exists \rho \sqsubseteq S$ such that $\rho' = \rho \cap X$, and $\delta(\rho') = \delta(\rho \cap X) = \delta(\rho) + \delta(X) - \delta(\rho \cup X)$ (by the modularity property, Lemma 5). $\delta(X)$ is the smallest possible value for δ, therefore $\delta(X) - \delta(\rho \cup X) \leq 0$ and $\delta(\rho') \leq \delta(\rho) \leq bg(S)$.
- else if $(S \cap X) \sqsubseteq \rho'$, then let $\rho = \text{rem}_X(\rho')$. We have

$$\delta(\rho') = \delta(\rho \cup X) = \delta(\rho) + \underbrace{\delta(X) - \delta(\rho \cap X)}_{\leq 0} \leq bg(S)$$

Let us now prove that $\forall i \in [1 \dots \alpha(G) - |L|]$, $\text{pref}_{-i}(S') \leq \text{pref}_{-i}(S)$, and in the case of equality $\ell_{S'}(-i) \leq \ell_S(-i)$. Let us first note that, since $\delta(X)$ reaches the minimum possible value for δ over all licit subsets, we have $\forall \sigma \sqsubseteq (S \setminus X)$, $\delta(\sigma) \geq 0$ (otherwise, $\sigma \cup X$ would be a licit subset with $\delta(\sigma \cup X) < \delta(X)$). In addition, remark that since $\delta(X) = |L| - \alpha(G)$, $\text{Pref}_{-i}(S') \sqsubseteq S \cap X$. Given these elements, let $\rho' \sqsubseteq \text{Pref}_{-i}(S')$, and $\rho \sqsubseteq S$ the smallest prefix of S such that $\rho' = \rho \cap X$. We have $\delta(\rho) = \delta(\rho') + \underbrace{\delta(\rho \setminus X)}_{\geq 0}$, so $\delta(\rho') \leq \delta(\rho)$. It simply remains to show that $\rho \sqsubseteq \text{Pref}_{-i}(S)$ to get $\delta(\rho') \leq \text{pref}_{-i}(S)$. Let therefore τ be a strict prefix of ρ. We indeed have $\delta(\tau) = \underbrace{\delta(\tau \cap X)}_{> -i} + \underbrace{\delta(\tau \setminus X)}_{\geq 0} > -i$. Therefore overall $\text{pref}_{-i}(S') = \max_{\rho' \sqsubseteq \text{Pref}_{-i}(S')} \delta(\rho') \leq \text{pref}_{-i}(S)$. In addition, $\delta(\tau) > -i \; \forall \tau \sqsubseteq \rho$ with $\rho \neq \tau$ implies, when $\text{pref}_{-i}(S) < +\infty$, $\ell_S(-i) \geq \ell_{S'}(-i)$.

As for $\text{suff}_{-i}(S')$, we have $\text{suff}_{-i}(S') = \text{pref}_{-i}(\overleftarrow{S'})$. By Property 6, $Y = G \setminus X$ is a separator for $G_{R \to L}$, and $\overleftarrow{S'} = (\overleftarrow{S} \cap Y) \cdot (\overleftarrow{S} \setminus Y)$. Per the arguments above, $\text{pref}_{-i}(\overleftarrow{S'}) \leq \text{pref}_{-i}(\overleftarrow{S})$ and $\ell_{\overleftarrow{S'}}(-i) \leq \ell_{\overleftarrow{S}}(-i)$. Overall, $\text{suff}_{-i}(S') = \text{pref}_{-i}(\overleftarrow{S'}) \leq \text{pref}_{-i}(\overleftarrow{S}) = \text{suff}_{-i}(S)$, and in the case of equality, $r_{S'}(-i) \leq r_S(i)$. □

Therefore, if G allows for a mixed independent set, any optimal schedule can be assumed to go through this independent set. A schedule that reaches IS cardinality $\alpha(G)$ is said to be *simple*. Lemma 6 thus yields the following:

Corollary 5. *Any graph G has a* simple *optimal schedule.*

Corollary 6. *For any schedule S of a bipartite graph G, $\exists S'$* simple *such that $S' \preccurlyeq S$*

Proof. Either G does not allow for a mixed MIS, in which case $\alpha(G) = \max(|L|, |R|)$ and any schedule is simple, or G allows for a mixed MIS and we can apply Lemma 5.4.2 to S. □

This separation result will be used in Section 5.5. We now turn to another algorithmic building block, that is a *merge* procedure for combining solutions for disjoint graphs into a global optimal.

5.4.3 Merge Procedure

Merging problem. Given two independent graphs G_1 and G_2, and two optimal orderings S_1 and S_2, it is natural to ask whether it is always sufficient to simply *interlace* S_1 and S_2 to get an optimal solution G, or if a *rearrangement* of S_1 and/or S_2 may be required.

In this section, we answer this question by showing that not only is rearranging is necessary in some cases, but figuring out the optimal rearrangement is NP-hard (Lemma 7). However, we also show that interlacing is enough when the two input schedules are in *canonical form*, as defined in Definition 32. The merging procedure can then be done in linear time, as shown in Theorem 20.

Related work. A similar merge procedure had already been designed in [207] for the cumulative cost-optimal scheduling problem, of which Bipartite Independent Set Reconfiguration is an instance. However, the corresponding "canonical form", *strictly-optimal schedules*, was not adaptable to algorithmic application, described in Section 5.6. The unpublished pre-prints [212, 205] also claimed to derive a merge procedure for the same problem. However, the merge ("combine") procedure of [212] relies on an unproven Observation (Observation 5.5 in [212]). Even if this Observation was correct, the merge procedure presented here achieves a better (linear) complexity. As for [205], it is unclear (in a similar fashion as [207]) how it could adapt to our algorithmic application.

Lemma 7. *Given two bipartite graphs G_1 and G_2, S_1 and S_2 optimal schedules for G_1 and G_2 respectively, and an integer k, the problem of deciding whether $bg(G_1 \cup G_2) \leq k$ is* `NP-hard`.

Proof. We prove the NP-hardness by reduction from the barrier problem. Given therefore a bipartite graph G with n vertices and an integer k', we build G_1, G_2, S_1, S_2 and k as follows. G_1 consists of G augmented with a $(n+1, n+1)$-biclique B_1, such that the $n+1$ left vertices of B_1 are dependencies of all the right vertices of G. Additionally, one left vertex b_1 is added as a dependency of all the right vertices of G. G_2 consists of a single biclique with $n+1$ left vertices and $k'+1$ right vertices.

Let S_1 start with B_1, followed by b_1, and a simplistic schedule for G, consisting of all the left vertices of G followed by all the right vertices of G. As B_1 is a biclique, $bg(G_1) \geq n+1$ and S_1 is optimal. As for S_2, the only possible schedule consists of all its left vertices followed by all of its right vertices.

We will now show that $bg(G_1 \cup G_2) \leq n+1$ if and only if $bg(G) \leq k'$. To start with, if G admits a schedule S with $bg(S) \leq k'$, then $B_1 \cdot B_2 \cdot b_1 \cdot S$ is a $n+1$-schedule for $G_1 \cup G_2$ (with B_1, B_2 in any order in which all left vertices are before the right vertices) and $bg(G_1 \cup G_2) \leq n+1$.

In the other direction, if $bg(G_1 \cup G_2) \leq n+1$, then B_1 is necessarily scheduled first, as scheduling any part of B_2 before B_1 would increase the baseline by a strictly positive amount and yield an overall budget $> n+1$. Likewise, scheduling any part of $\{b_1\} \cup G$ before B_2 would yield a barrier $> n+1$. Therefore, an optimal schedule is necessary of the same form as before $B_1 \cdot B_2 \cdot b_1 \cdot S$ for some schedule S of G. If it has barrier $\leq n+1$, then necessarily $bg(S) \leq k'$, concluding the proof.

□

lumps: small non-breakable units. The merge procedure will exploit an interesting aspect of canonical solutions: they start with the shortest way, if possible, to get the budget below the original baseline. This is illustrated by the following lemma.

Definition 33. A licit subset X of a graph G is a *lump* if:

1. $\delta(X) < 0$
2. $\forall X' \subsetneq X$ licit, we have $\delta(X') \geq 0$

Moreover, a lump X is *harmless* if its budget $bg(X)$ is minimal among all lumps, and $|X|$ is minimal among all lumps with this budget.

In other words, when ordering lumps according to the lexicographic order over $(bg(X), |X|)$, a harmless lump is an absolute minimum.

Property 7. *A lump X induces a connected subgraph of G.*

Proof. Suppose $G[X]$ is composed of two components induced by $X_1 \subsetneq X$ and $X_2 \subsetneq X$. Both are licit, so by the definition of a lump, $\delta(X_1) \geq 0$ and $\delta(X_2) \geq 0$. However $\delta(X) = \delta(X_1) + \delta(X_2) < 0$, hence a contradiction. □

Lemma 8. *If S is a simple schedule for a bipartite graph G and X is a lump of G admitting an optimal schedule S_X such that $bg(S_X) \leq \min(pref_{-1}(S), bg(S))$, then $S' := S_X \cdot (S \setminus X)$ is preferable to S. It is moreover strictly preferable if either $bg(S_X) < pref_{-1}(S)$ or $|X| < |Pref_{-1}(S)|$.*

Proof. We first recall the following notation: given a prefix ρ' of S' with $S_X \sqsubseteq \rho'$, we write $\text{rem}_X(\rho')$ for the smallest prefix ρ of S such that $V(\rho \cup X) = V(\rho')$. Conversely, we also define for a prefix ρ of S $\text{add}_X(\rho)$ as the smallest prefix ρ' of S such that $V(\rho \cup X) = V(\rho')$. Note that add_X and rem_X are monotonous under the prefix relation. These definitions are illustrated in Figure 5.4. Note also that for any prefix ρ of S, $\rho'' = \text{rem}_X(\text{add}_X(\rho))$ is the smallest prefix σ of S such that $V(\sigma \cup X) = V(\rho \cup X)$, so in particular $\rho'' \sqsubseteq \rho$. For any such pair ρ, ρ' with $V(\rho \cup X) = V(\rho')$, we show that $\delta(\rho') \leq \delta(\rho)$. Indeed, $\delta(\rho') = \delta(\rho \cup X) = \delta(\rho) + \delta(X) - \delta(\rho \cap X)$ (per the modularity of δ, Lemma 5). Since $\rho \cap X$ is licit (by Property 5) by the definition of lump we have $\delta(\rho \cap X) \geq \delta(X)$ and $\delta(\rho') \leq \delta(\rho)$.

We can now show that S' is preferable to S, starting with $bg(S') \leq bg(S)$. Consider a prefix ρ' of S'. If $\rho' \sqsubseteq S_X$, then $\delta(\rho') \leq bg(X) \leq bg(S)$. Otherwise, $S_X \sqsubseteq \rho'$, and $\delta(\rho') \leq \delta(\text{rem}_X(\rho')) \leq bg(S)$.

Then, we have $\text{pref}_{-1}(S') = bg(S_X) \leq \text{pref}_{-1}(S)$ by assumption. To continue, for i such that $\text{pref}_{-i}(S) < +\infty$, we have $\text{pref}_{-i}(S') \sqsubseteq \text{add}_X(\text{pref}_{-i}(S))$ (indeed, $\delta(\text{add}_X(\text{pref}_{-i}(S))) \leq \delta(\text{pref}_{-i}(S)) = -i$, so $\text{add}_X(\text{pref}_{-i}(S))$ is some prefix, not necessarily smallest, of S' with balance no more than $-i$). Thus, for any ρ', $S_X \sqsubseteq \rho' \sqsubseteq \text{Pref}_{-i}(S')$, we have $\delta(\rho') \leq \delta(\text{rem}_X(\rho'))$ and

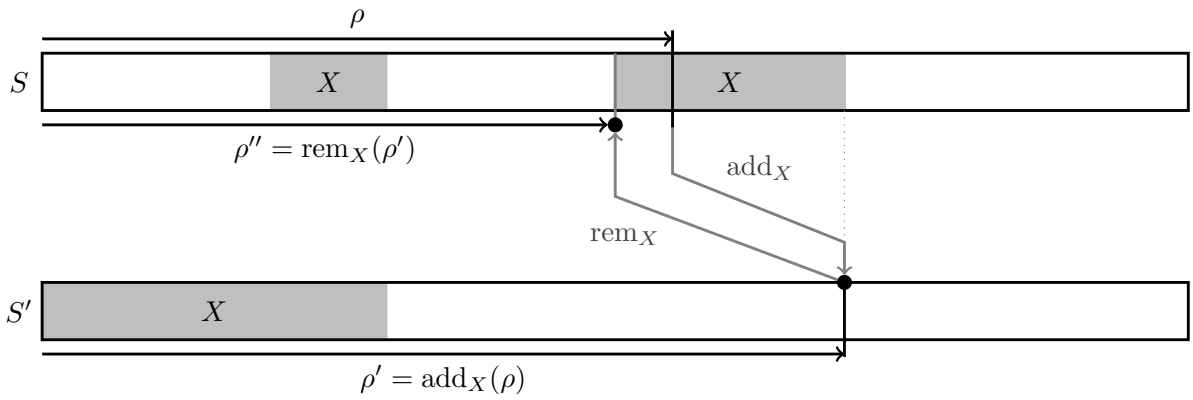


Figure 5.4: Illustration of the definitions of add_X and rem_X, used in the proofs of Lemmas 5.4.2 (only add_X),8 and 10. They apply to a typical situation encountered in these Lemmas: a schedule S is shuffled so that a licit set X is processed first. The new schedule S' is valid since X is licit. The purpose of add_X and rem_X is then to draw connections between prefixes of S and corresponding prefixes in S', in order to infer bounds on $bg(S')$, $\mathrm{pref}_{-i}(S')$ or $\mathrm{suff}_{-i}(S')$.

$\mathrm{rem}_X(\rho')$ is a prefix of $\mathrm{rem}_X(\mathrm{add}_X(\mathrm{Pref}_{-i}(S')))$ so $\delta(\mathrm{rem}_X(\rho')) \leq \mathrm{pref}_{-i}(S')$. Overall, any prefix of $\mathrm{pref}_{-i}(S')$ has balance at most $\max(bg(S_X), \mathrm{pref}_{-i}(S))$ and $bg(S_X) \leq \mathrm{pref}\_1(S') \leq \mathrm{pref}_{-i}(S')$ so $\mathrm{pref}_{-i}(S') \leq \mathrm{pref}_{-i}(S)$.

To finish, consider if it exists an i such that $\mathrm{suff}_{-i}(S) < +\infty$. By the existence of a licit subset X with $\delta(X) < 0$ and of $\mathrm{Suff}_{-i}(S)$, we have the existence of a mixed maximum independent set in G. As S is simple, it does reach this minimal balance, and there is $\rho \sqsubseteq S$ such that $\delta(\rho) = |L| - \alpha(G)$, the lowest possible value for δ. A useful consequence is that we must have $X \subseteq \rho$, as otherwise, $\delta(X \cap \rho) \geq 0$ (by definition of lump), and when reshuffling we would obtain $\delta(\rho \cup X) = \delta(\rho) + \delta(X) - \delta(X \cap \rho) \leq \delta(\rho) - 1$, which is not possible by minimality of $\delta(\rho)$ over licit subsets. As a consequence $\forall i$, $\mathrm{Suff}_{-i}(S') = \mathrm{Suff}_{-i}(S)$ and $\mathrm{suff}_{-i}(S') = \mathrm{suff}_{-i}(S)$.

Overall, $S' = S_X \cdot S \setminus X$ is indeed preferable to S. Note also that if $bg(S_X) < \mathrm{pref}_{-1}(S)$, then $\mathrm{pref}_{-1}(S') < \mathrm{pref}_{-1}(S)$, and if $|X| < |\mathrm{Pref}_{-1}(S)\|$, then $\ell_{S'}(-1) < \ell_S(-1)$: in both cases, S' is strictly preferable to S.

□

Canonical solutions start with lumps. Lemma 8 is akin to the "commitment lemma" of [211] and Lemma 4.6 of [212]. However, in this paper, we link this result to newly-introduced notions of preferability (Definition 31) and canonicity (Definition 32). This is the case in particular of Lemma 10. It relies itself on the following existence result for lumps, essentially saying that a balanced licit set can always be reduced to a lump.

Lemma 9. *If X is a licit set with $\delta(X) < 0$, then either X is a lump or there is a lump $Y \subsetneq X$ with $bg(Y) \leq bg(X)$.*

Proof. The proof is by induction on $|X|$. Pick $X' \subseteq X$ licit and minimal-by-inclusion under the condition $\delta(X') < 0$. Per the minimality criteria, X' is a lump. If $bg(X') \leq bg(X)$, we are done.

If $bg(X') > bg(X)$, consider S an optimal schedule for X, and ρ the largest prefix of S such that $\delta(\rho \cap X') = bg(X)+1$: such a prefix exists, since any schedule of X' (and in particular $S \cap X'$) reaches balance $bg(X)+1$ at some point. Let also σ denote the suffix of S corresponding to ρ, i.e. such that $S = \rho \cdot \sigma$. Consider now the set $Y' = \rho \setminus X'$. We have $\delta(\rho) = \delta(Y') + \delta(\rho \cap X') = \delta(Y') + (bg(X)+1)$ and $\delta(\rho) \leq bg(X)$ so $\delta(Y') \leq -1$. Overall, $Y' \cup X'$ is a licit set ($= \rho \cup X'$) with balance ≤ -2. In addition, $bg(Y' \cup X') \leq bg(X)$ with the schedule $S' = \rho \cdot (\sigma \cap X')$. The definitions of ρ, X', Y', σ and their relations to one another are illustrated in Figure 5.5

Then, $Z = V(\text{Pref}_{-1}(S'))$ yields a licit subset with $\delta(Z) < 0$ and $Z \subsetneq Y' \cup X'$ (as $\delta(Z) = -1$ while $\delta(S') = -2$). Therefore $|Z| < |X|$, and we can apply the induction hypothesis to it: Z is either a lump or contains one. In either case, there is a lump strictly included in X. □

Lemma 10. *If G is a bipartite graph for which there exists an optimal schedule S with $pref_{-1}(S) < +\infty$, then a canonical schedule S_C for G necessarily starts with a harmless lump $X = V(Pref_{-1}(S_C))$.*

Proof. We first show that $X = V(\text{Pref}_{-1}(S_C))$ is indeed a lump. By Lemma 9, since X is licit and $\delta(X) < 0$, either it is a lump or there exists a lump $Y \subseteq X$ with $|Y| < |X|$ and $bg(Y) \leq bg(X) = \text{pref}_{-1}(S_C) \leq bg(G)$. Applying Lemma 8 to S_C and Y would yield a schedule strictly preferable to S_C, which is not possible. X is therefore indeed a lump.

Finally, X is indeed harmless. Otherwise, there would exist a lump Y such that $bg(Y) < bg(X)$ or $|Y| < |X|$ if $bg(X) = bg(Y)$. By Lemma 8, $S_Y \cdot (S \setminus Y)$, with S_Y an optimal schedule for Y would be strictly preferable to the canonical schedule S, which is not possible. □

Harmless lumps may start canonical schedule. The merge procedure will need both the fact that (1) one can find harmless lumps at the beginning of canonical schedules (Lemma 10) and (2) one can put any harmless lump at the beginning of a canonical schedule. This latter point is the subject of the following Lemma. It will also allow to show that a canonical schedule always exists for any instance of BISR.

Lemma 11. *If X is a harmless lump of G with optimal schedule S_X, and S_C a canonical schedule for $G \setminus X$, then $S_X \cdot S_C$ is a canonical schedule for G*

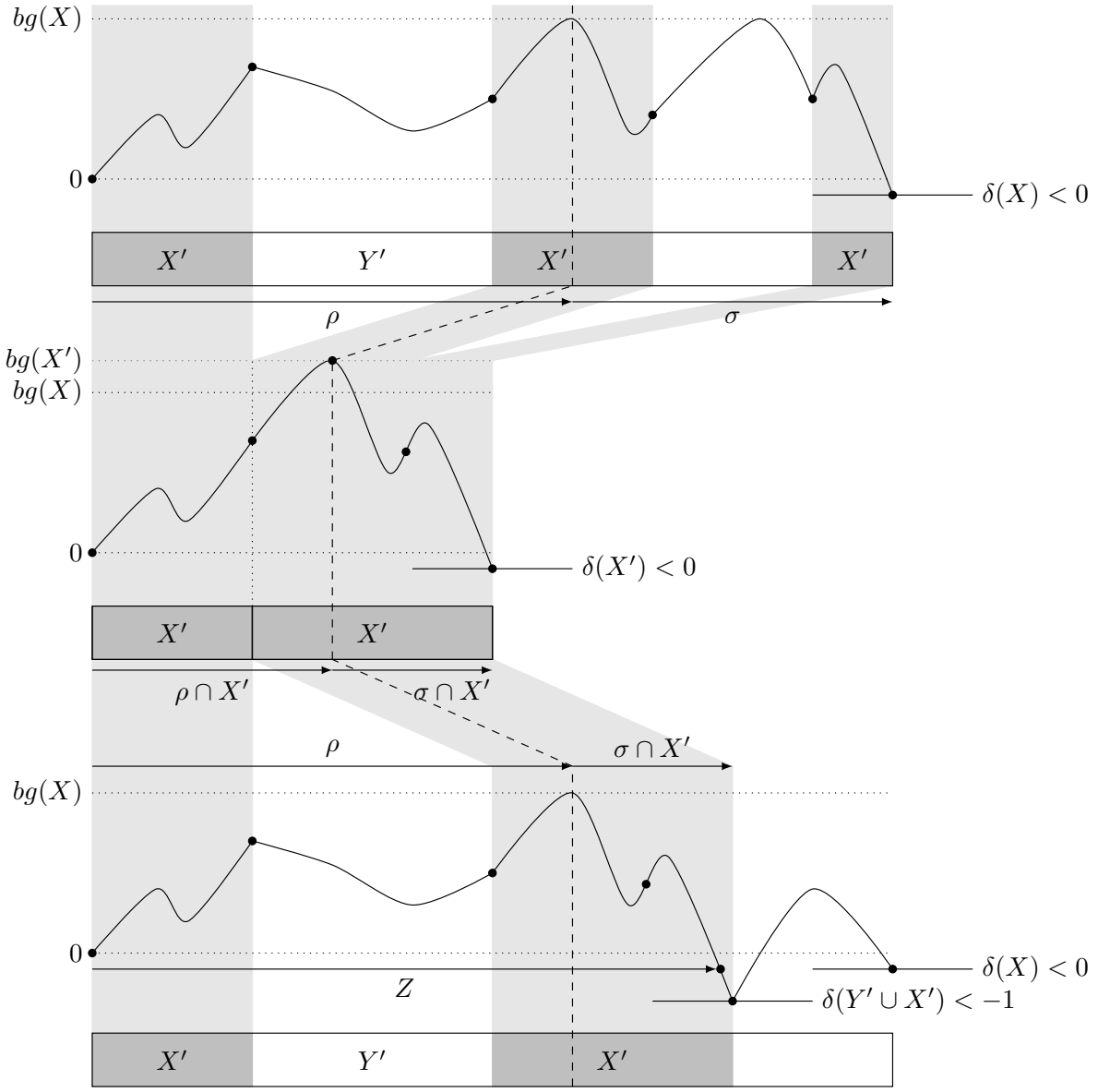


Figure 5.5: Illustration of the objects used in the proof of Lemma 9. The purpose is to show that, given a licit set X such that $\delta(X) < 0$, if it is not a lump itself, a smaller licit set with negative balance $Z \subsetneq X$ can be found. The induction hypothesis can then be applied to Z to show that it contains a lump.

Proof. Let S_G be a schedule for G. We will prove that $S_X \cdot S_C \preccurlyeq S_G$.

Let us first apply Corollary 6 to get S'_G simple and preferable to S_G. We can then apply Lemma 8 to S'_G and X. The only criteria to verify is $bg(S_X) \leq \min(\text{pref}_{-1}(S'_G), bg(S'_G))$. Since X is a licit subset with $\delta(X) < 0$, $\alpha(G) \geq |I(X)| \geq |L|$, and since S'_G is simple, $\text{pref}_{-1}(S'_G) < +\infty$. By Lemma 9, there exists a lump Y in $V(\text{Pref}_{-1}(S'_G))$ with $bg(Y) \leq \text{pref}_{-1}(S'_G)$. As X is a harmless lump, $bg(S_X) \leq bg(Y) \leq \text{pref}_{-1}(S'_G) = \min(bg(S'_G), \text{pref}_{-1}(S'_G))$. In addition, if $bg(S_X) = bg(Y)$, we know that $|S_X| \leq |Y|$.

By Lemma 8, $S_X \cdot (S'_G \setminus X) \preccurlyeq S'_G \preccurlyeq S_G$. Let us know replace $S'_G \setminus X$ by S_C, a canonical schedule for $G \setminus X$, and show that we obtain a schedule preferable to $S_X \cdot (S'_G \setminus X)$. We have $\text{Pref}_{-1}(S_X \cdot S_C) = S_X = \text{Pref}_{-1}(S_X \cdot (S'_G \setminus X))$, and $\forall i \in [-2, \ldots, |L| - \alpha(G)]$:

$$\text{Pref}_{-i}(S_X \cdot S_C) = S_X \cdot \text{Pref}_{-i+1}(S_C)$$

and

$$\text{Pref}_{-i}(S_X \cdot (S'_G \setminus X)) = S_X \cdot \text{Pref}_{-i+1}(S'_G \setminus X)$$

Given that S_C is canonical for $G \setminus X$, it is preferable to $S'_G \setminus X$. Therefore $\forall i, \text{pref}_{-i}(S_C) \leq \text{pref}_{-i}(S'_G \setminus X)$, with $\text{Pref}_{-i}(S_C)$ shorter in case of equality.

As for $\text{Suff}_{-i}(S_X \cdot S_C)$ for any $i \in [-1, \ldots, |R| - \alpha(G)]$, we have from the definition of a lump

$$\text{Suff}_{-i}(S_X \cdot S_C) = \text{Suff}_{-i}(S_C)$$

and

$$\text{Suff}_{-i}(S_X \cdot (S'_G \setminus X)) = \text{Suff}_{-i}(S'_G \setminus X)$$

As above, since S_C is canonical, $\text{suff}_{-i}(S_C) \leq \text{suff}_{-i}(S'_G \setminus X)$ with $\text{Suff}_{-i}(S_C)$ shorter in case of equality, and the same goes for $S_X \cdot S_C$ and $S_X \cdot (S'_G \setminus X)$.

Finally,

$$\begin{aligned} bg(S_X \cdot S_C) &= \max\left(bg(S_X), -1 + bg(S_C)\right) \\ &\leq \max\left(bg(S_X), -1 + bg(S'_G \setminus X)\right) \\ &= bg(S_X \cdot S'_G \setminus X) \end{aligned}$$

Overall, $S_X \cdot S_C \preccurlyeq S_G$ and is therefore a canonical schedule. □

Corollary 7 (existence of a canonical solution)**.** *There always exists a canonical schedule for a given bipartite graph G*

Proof. Consider S a simple optimal schedule for G. If $\text{pref}_{-1}(S) < +\infty$, then $\exists p \sqsubseteq S$ such that $\delta(p) < 0$. By Lemma 9, either p is a lump or it contains a lump X with $bg(X) \leq bg(Y) \leq \text{pref}_{-1}(S) \leq bg(S)$. Let us pick X harmless. By Lemma 11 with S_X an optimal schedule for X, and by induction, we get a canonical schedule for G.

If $\text{suff}_{-1}(S) < +\infty$, then $\text{pref}_{-1}(\overleftarrow{S}) < +\infty$. With the analysis above, we get a canonical schedule for $G_{R\to L}$ which can be reversed into a canonical schedule for G.

If none of the cases above apply, and because S is simple, then necessarily $|L| = |R|$ and G does not allow for any mixed-MIS. In that case, $\text{suff}_{-i}(S) = \text{pref}_{-i}(S) = +\infty\ \forall i$ and any optimal schedule is canonical. □

Lemma 12. *Let (G_1, G_2) be two disjoint bipartite graphs and (S_1, S_2) canonical solutions for (G_1, G_2) respectively. If $\text{pref}_{-1}(S_1)$, $\text{pref}_{-1}(S_2)$, $\text{suff}_{-1}(S_1)$ and $\text{suff}_{-1}(S_2)$ are all equal to $+\infty$ then both schedules $S_1 \cdot S_2$ and $S_2 \cdot S_1$ are optimal and canonical.*

Proof. First, let us note that $\text{pref}_{-1}(S_1) = \text{pref}_{-1}(S_2) = \text{suff}_{-1}(S_1) = \text{suff}_{-1}(S_2) = +\infty$ implies $\delta(G_1) = \delta(G_2) = 0$, by definition of pref_{-1} and suff_{-1}.

Then, G_1 (resp. G_2) cannot allow for a mixed independent set of size $> |L_1| = |R_1|$ (resp. $|L_2| = |R_2|$), as it would imply by Lemma 6 the existence of an optimal schedule with $\text{pref}_{-1}(S_1) < +\infty$ (resp, $\text{pref}_{-1}(S_2) < +\infty$). As a consequence, The maximum independent sets of G are exactly $L_1 \cup L_2$, $L_1 \cup R_2$, $R_1 \cup L_2$ and $R_1 \cup R_2$.

Both $L_1 \cup R_2$ and $R_1 \cup L_2$ are mixed maximum independent sets. Consider therefore an optimal (and therefore, canonical) schedule S for G. By Lemma 6 both $S_1 \cdot (S \setminus G_1)$ and $S_2 \cdot (S \setminus G_2)$ are preferable to S and canonical. Replacing $S \setminus G_1$ by S_2 and $S \setminus G_2$ by S_1 does not increase the budget, which is equal to $\max(bg(S_1), bg(S_2))$. Since the budget is the only criteria for preferability in this case, both $S_1 \cdot S_2$ and $S_2 \cdot S_1$ are canonical. □

Merge theorem. The following Theorem is the main result of this section, and essentially states that a canonical (and therefore optimal) solution for $G = G_1 \cup G_2$ can be obtained by interleaving canonical solutions for two disjoint graphs G_1 and G_2. This suggests the merging procedure implemented by Algorithm 3, where $\text{Pref}_{-1}(S_1), \text{Pref}_{-1}(S_2), \text{Suff}_{-1}(S_1)$ and $\text{Suff}_{-1}(S_2)$ are treated as "canonical blocks", i.e. are not broken up in the interleaving process.

Theorem 20 (merge algorithm)**.** *If G_1 and G_2 are two disjoint bipartite graphs and S_1 and S_2 two canonical solutions for G_1 and G_2 respectively, then Algorithm 3 yields a canonical solution for G in $O(|S_1| + |S_2|)$.*

Proof. **Run-time:** Given a schedule S, $\text{pref}_{-1}(S)$ and $\text{suff}_{-1}(S)$ and (if they exist) $\text{Pref}_{-1}(S), \text{Suff}_{-1}(S)$ can be computed in $O(|S|)$ by a simple iteration over S that keeps track of the budget.

Correctness: We prove the correctness by induction, with the base case being when $S_1 = \emptyset$ or $S_2 = \emptyset$.

In the general case, if $\min(\text{pref}_{-1}(S_1), \text{pref}_{-1}(S_2)) < +\infty$, consider (w.l.o.g) that $(\text{pref}_{-1}(S_1), \ell_{S_1}(-1)) \leq_{\text{lex}} (\text{pref}_{-1}(S_2), \ell_{S_2}(-1))$, with $\leq_{\text{lex}}$ denoting the lexico-graphic order.

By Lemma 10 applied to G_1 and S_1, $\text{Pref}_{-1}(S_1)$ is a harmless lump of G_1. It is therefore a lump of G. Let us prove it is also harmless in G. To that end, note that lumps are connected (Property 7), so a lump of G is either a lump of G_1 or G_2. Therefore $(\text{pref}_{-1}(S_1), \ell_{S_1}(-1)) \leq_{\text{lex}} (\text{pref}_{-1}(S_2), \ell_{S_2}(-1))$ indeed implies that $\text{Pref}_{-1}(S_1)$ is of minimal budget among lumps of G, and shorter than $\text{Pref}_{-1}(S_2)$ in case of budget equality.

Since by the induction hypothesis $\textsc{Merge}(S_1 \setminus \text{Pref}_{-1}(S_1), S_2)$ is canonical, $\text{Pref}_{-1}(S_1) \cdot \textsc{Merge}(S_1 \setminus \text{Pref}_{-1}(S_1), S_2)$ is canonical by Lemma 11

The case $\min(\text{suff}_{-1}(S_1), \text{suff}_{-1}(S_2)) < +\infty$ (lines 10-16) is treated with the same arguments, given the symmetry of pref and suff when inverting L and R, namely $\forall S\ \text{suff}_{-1}(S) = \text{pref}_{-1}(\overleftarrow{S})$.

As for the justification of the concatenation if none of the conditions above apply, it is brought by Lemma 12.

□

Algorithm 3 Merge procedure for canonical schedules. $\leq_{\text{lex}}$ denotes the lexicographic order, applied here to couples of integers.

Input: S_1, S_2 canonical solutions for G_1, G_2 (disjoint graphs)
Output: a canonical solution S for $G = G_1 \cup G_2$

1: **function** $\textsc{Merge}(S_1, S_2)$:
2: ▷ If first or last canonical blocks exist: recurse
3: **if** $\min(\text{pref}_{-1}(S_1), \text{pref}_{-1}(S_2)) < +\infty$ **then**
4: **if** $(\text{pref}_{-1}(S_1), \ell_{S_1}(-1)) \leq_{\text{lex}} (\text{pref}_{-1}(S_2), \ell_{S_2}(-1))$ **then**
5: **return** $\text{Pref}_{-1}(S_1) \cdot \textsc{Merge}(S_1 \setminus \text{Pref}_{-1}(S), S_2)$
6: **else**
7: **return** $\text{Pref}_{-1}(S_2) \cdot \textsc{Merge}(S_1, S_2 \setminus \text{Pref}_{-1}(S_2))$
8: **end if**
9: **end if**
10: **if** $\min(\text{suff}_{-1}(S_1), \text{suff}_{-1}(S_2)) < +\infty$ **then**
11: **if** $(\text{suff}_{-1}(S_1), r_{S_1}(-1)) \leq_{\text{lex}} (\text{suff}_{-1}(S_2), r_{S_2}(-1))$ **then**
12: **return** $\textsc{Merge}(S_1 \setminus \text{Suff}_{-1}(S_1), S_2) \cdot \text{Suff}_{-1}(S_1)$
13: **else**
14: **return** $\textsc{Merge}(S_1, S_2 \setminus \text{Suff}_{-1}(S_2)) \cdot \text{Suff}_{-1}(S_2)$
15: **end if**
16: **end if**
17: ▷ If no first or last canonical block exist: simply concatenate
18: **return** $S_1 \cdot S_2$
19: **end function**

5.5 Parameterized algorithms for bipartite independent set reconfiguration

In this section, we apply the technical results of the previous section to the design of XP algorithms for BISR. For example, Lemma 6 is used to formulate a $O(n^2)$-space, $O(n^{2\rho})$-time algorithm for BISR, described in Section 5.5.1. As for Lemma 20, it allows to build a $O(n^{\Phi+1})$ algorithm for BISR when restricted to *bipartite circle graphs*, described in Section 5.6. Bipartite circle graphs constitute a subcase of interest to RNA kinetics, as we shall see in more details in Section 5.7.

5.5.1 An XP algorithm in ρ

Overall strategy. Lemma 6 allows for a divide-and-conquer approach: if we identify a separator X in G, i.e. a licit subset of G such that $I(X)$ is a mixed independent set, then we may independently solve the problem of finding a ρ-realization from L to $I(X)$ and then from $I(X)$ to R. If no solution is found for one of them, then the converse of Lemma 6 implies that no ρ-realizations exists for G. The algorithm presented in this section is based on this approach.

Algorithm details. We present here a direct algorithm for Bipartite Independent Set Reconfiguration, detailed in Algorithm 4. The main function `Realize` is recursive. Its sub-calls arise either from a split with a mixed MIS I (in which case it is called on a smaller graph but with the same parameter), or from the loop over all possible starting points in the case where no separator is found (lines 13-18), in which case the parameter does reduce. The overall runtime is dominated by this loop, and is analyzed in Proposition 11 below.

Mixed MIS algorithm. The sub-routine allowing to find, if it exists, a maximum independent set intersecting both L and R is based on concepts from *matching theory* [213], namely the *Dulmage-Mendelsohn* decomposition [214, 213], as well as the decomposition of bipartite graphs with a perfect matching into *elementary subgraphs* [213](part 4.1). Its full details are described in the full version of the article.

Proposition 11. *Algorithm 4 runs in $O(|V|^{2\rho}\sqrt{|V|}|E|)$ time, while using $O(|V|^2)$ space, where ρ is the difference between the minimum allowed and maximum possible independent set size, along the reconfiguration.*

Proof. Let us start with space: throughout the algorithm, one needs only to maintain a description of G and related objects (independent set I, maximum matching M, associated directed graph $H(G, M)$) for which $O(|V|^2)$ is enough.

As for time, let $C(n_1, n_2, \rho)$ be the number of recursive calls of the function *Realize* of Algorithm 4 when initially called with $|L| = n_1$, $|R| = n_2$, and some value of ρ. We will show by induction that $C(n_1, n_2, \rho) \leq (n_1 + n_2)^{2\rho}$. Since each call involves one computation of a maximum matching, this will prove our result.

Algorithm 4 XP algorithm for Bipartite Independent Set Reconfiguration

Input: bipartite graph G (with sides L and R), integer ρ
Output: a ρ-realization for G, if it exists

```
1:  function Realize(G, ρ):
2:
3:      if ρ < 0 then return ⊥                                  ▷ // Terminal cases:
4:      end if
5:      if L ∪ R = ∅ then return ∅
6:      end if
7:      if ∃ℓ ∈ L s.t N(ℓ) = ∅ then return Realize(G \ {ℓ}, ρ − 1) · ℓ
8:      end if
9:      if ∃r ∈ R s.t N(r) = ∅ then return r · Realize(G \ {r}, ρ − 1)
10:     end if
11:
12:     I = MixedMIS(G)                                         ▷ // Trying to find a separator
13:     if I ≠⊥ then
14:         S = (L \ I) ∪ (R ∩ I)
15:         return Realize(G[S], ρ)· Realize(G[V \ S], ρ)
16:     else
17:         for (ℓ, r) ∈ L × R do                               ▷ // loop over all start-end possibilities
18:             if Realize(G \ {ℓ, r}, ρ − 1)≠⊥ then
19:                 return ℓ · Realize(G \ {ℓ, r}, ρ − 1) · r
20:             end if
21:         end for
22:     end if
23: end function
```

Given (n_1, n_2, ρ), suppose therefore that $\forall (n_1', n_2', \rho') \neq (n_1, n_2, \rho)$ with $n_1' \leq n_1, n_2' \leq n_2, \rho' \leq \rho$ we have $C(n_1', n_2', \rho') < (n_1' + n_2')^{2\rho'}$

1. If G allows for a mixed maximum independent set, the instance is split into two smaller instances, yielding $C(n_1, n_2, \rho) = C(n_1', n_2, \rho) + C(n_1'', n_2'', \rho)$ with $n_1' + n_1'' = n_1$ and $n_2 = n_2' + n_2''$. And $C(n_1, n_2, \rho) \leq \left((n_1' + n_2')^{2\rho} + (n_1'' + n_2'')^{2\rho}\right) \leq (n_1' + n_1'' + n_2' + n_2'')^{2\rho} \leq (n_1 + n_2)^{2\rho}$.

2. else, we have the following relation: $C(n_1, n_2, \rho) = n_1 n_2 \cdot C(n_1 - 1, n_2 - 1, \rho - 1)$. Which yields:

$$\begin{aligned} C(n_1, n_2, \rho) &= n_1 n_2 \cdot C(n_1 - 1, n_2 - 1, \rho - 1) \\ &\leq n^2 \cdot n^{2(\rho-1)} \quad \text{by induction hypothesis} \\ &\leq n^{2\rho} \end{aligned}$$

□

The exponential part ($O(n^{2\rho})$) of the worst case complexity of Algorithm 4 is in fact tight, as it is met with a complete bi-clique $K_{n,n}$ with sides of size n. Indeed, in this case, no mixed MIS is found in any of the recursive calls.

5.6 RNA case: bipartite circle graphs

5.6.1 RNA basics and arboricity parameter

RNA Energy Barrier **problem.** As touched upon in Chapter 1, RiboNucleic Acids (RNAs) are biopolymers composed of four possible nucleotides, which can therefore be represented as strings over an alphabet $\Sigma := \{\mathtt{A}, \mathtt{C}, \mathtt{G}, \mathtt{U}\}$. Importantly, these strings may *fold* on themselves to adopt one or several conformation(s), also called *structures*. For a string of length N, a conformation is typically described by a set S of base pairs (i, j), with $1 \leq i < j \leq N$. Then, a standard class of conformations to consider in RNA bioinformatics are *conflict-free secondary structures* (Definition 1.5), which are pairwise non-crossing ($\nexists (i, j), (k, l) \in S$ such that $i \leq k \leq j \leq l$, in particular, they involve distinct positions). Due to this non-crossing property, secondary structures are in bijection with *well-parenthesized* strings, as illustrated in Figure 5.6 (B).

Problem statement. In this section, we more precisely work on the problem of finding a reconfiguration pathway between two *secondary structures* (i.e conflict-free sets of base pairs). The reconfiguration may only involve secondary structures, and remain of energy as low as possible. We work with the simple energy model ($E_{\texttt{\#-bps}}$) consisting of the opposite of *number of base pairs* in a configuration ($-N_{bps}$). We recall here the statement of RNA Energy Barrier:

Problem (RNA Energy Barrier).
Input: *Secondary structures L and R; Energy barrier $k \in \mathbb{N}^+$*
Output: True *if there exists a sequence $S_0 \cdots S_\ell$ of secondary structures such that (1) $S_0 = L$ and $S_\ell = R$; (2) $|S_i| \geq |L| - k, \forall i \in [0, \ell]$ and (3) $|S_i \triangle S_{i+1}| = 1, \forall i \in [0, \ell - 1]$.* False *otherwise.*

Problem motivation. Since the number of secondary structures available to a given RNA grows exponentially with n, RNA energy landscapes are notoriously *rugged*, i.e. feature many local minima, and the folding process of an RNA from its *synthesis* to its theoretical final state (a thermodynamic equilibrium around low energy conformations) can be significantly slowed down. Consequently, some RNAs end up being degraded before reaching this final state. This observation motivates the study of RNA kinetics, which encompass all time-dependent aspects of the folding process. In particular, it is known (Arrhenius law) that the energy barrier is the dominant factor influencing the transition rate between two structures, with an exponential dependence.

BISR **on circle graphs.** Two arcs (i, j) and (k, l) are said to be in *conflict* or *crossing* if $i \leq k \leq j \leq l$ or $k \leq i \leq l \leq j$ (i.e., when there is not one of them nested in the other). It simply means that they cannot be both present at the same time in an RNA secondary structure.

To capture this constraint in reconfiguring RNA secondary structures, we define the *conflict graph* $G(L, R)$ as having $L \cup R$ as vertices, and an edge connecting two arcs if they are in conflict. L and R being two valid secondary structures, the graph is bipartite. More generally, a valid secondary structure is then an *independent set* of $G(L, R)$. Reconfiguring L into R while minimizing energy along the way then consists in solving BISR on $G(L, R)$. The following proposition characterizes the set of bipartite graphs that emerge from this construction, as *bipartite circle graph*. A circle graph is an intersection-graph of chords of a circle.

Proposition 12. *The* RNA energy barrier *problem as defined above is* BISR *restricted to bipartite circle graphs*

Proof. Given L and R two well-nested arc sets (i.e., two RNA secondary structures) over $[1 \dots n]$, denoted $L = \{(l_i, r_i)\}$ and $R = \{b_j, e_j\}$. Consider a circle with n regularly-spaced positions on it, and the set of chords $L \cup R$. The associated circle graph (chord-intersection graph) is exactly the conflict-graph $G(L, R)$. There is then an exact correspondence between independent sets of $G(L, R)$ and valid secondary structures composed of arcs from L and R.

Conversely, given a bipartite circle graph, its two sides L and R yield two well-nested arc sets that can be seen as RNA secondary structures. The correspondence is highlighted on Figure 5.6. □

Arboricity (Φ). Given an RNA secondary structure S (a set of well-nested base-pairs) the arboricity Φ of S is the number of "terminal" base-pairs, i.e. the number of base-pairs that do not contain any nested base-pair. A formal definition is given below.

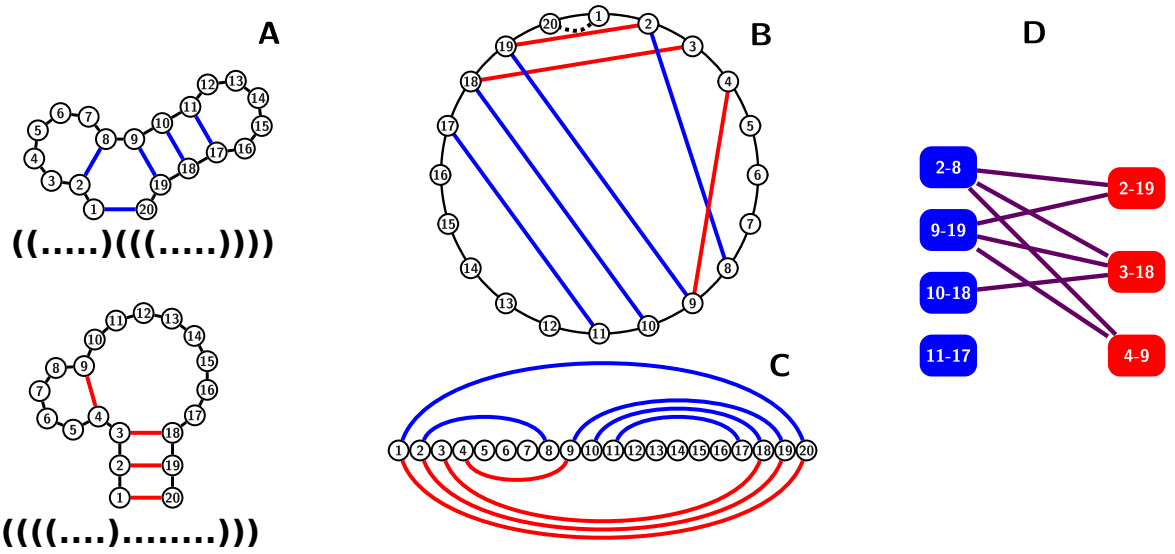


Figure 5.6: Conflict bipartite graph (D) associated with an instance of the RNA Energy-Barrier problem, consisting of an initial (A) and final (B) structure, both represented as an arc-annotated sequence (C). The sequence of valid secondary structures, achieving minimum energy barrier can be obtained from the solution given in Figure 5.6.

Notations for base-pair relations. We use here Notation 1 for base-pair relations, which we recall here. Given two base-pairs (i, j) and (k, l), we write $(k, l) \subset (i, j)$ if (k, l) is *nested* in (i, j), i.e. if $i < k < l < j$. One may see this notation as "the interval $[k, l]$ is a proper subset of the interval $[i, j]$". For two non-conflicting base-pairs, if no one of them is nested in the other, we write $(i, j) \parallel (k, l)$, which means either $i < j < k < l$ or $k < l < i < j$.

Definition 34. The arboricity $\Phi(S)$ of a set of well-nested base-pairs is:

$$\Phi(S) = |\{(i, j) \in S \mid \nexists (k, l) \in S \text{ with } (k, l) \subset (i, j)\}|$$

When seeing an RNA secondary structure as a set of well-parenthesized strings (Figure 5.6.A for instance), it is the number of matching opening/closing parenthesis symbols that only have dots between them.

Separating inside and outside sub-instances. Given L and R two well-nested arc sets, and $\ell = (i, j)$ an element of L. ℓ defines naturally "inside" and "outside" sub-instances in L and R, that are only connected through $N(\ell)$, the elements of R in conflict with ℓ. Formally these "inside" and "outside" sub-instances are $(L^\ell_{\text{IN}}, R^\ell_{\text{IN}})$ and $(L^\ell_{\text{OUT}}, R^\ell_{\text{OUT}})$ with $L^\ell_{\text{IN}} = \{\ell' \in L \mid \ell' \subset \ell\}$, $R^\ell_{\text{IN}} = \{\ell' \in R \mid \ell' \subset \ell\}$, $L^\ell_{\text{OUT}} = \{\ell' \in L \mid \ell \subset \ell' \text{ or } \ell \parallel \ell'\}$ and $R^\ell_{\text{OUT}} = \{\ell' \in R \mid \ell \subset \ell' \text{ or } \ell \parallel \ell'\}$ Note that

$\{\ell\}, L^{\ell}_{\text{IN}}, L^{\ell}_{\text{OUT}}$ form a partition of L, and $N(\ell), R^{\ell}_{\text{IN}}, R^{\ell}_{\text{OUT}}$ form a partition of R.

5.6.2 An XP algorithm for Φ

Dynamic programming table. The algorithm we present in this Section (Algorithm 5) is based on dynamic programming, using a memorization strategy. There is therefore a table in which solutions to partial instances are stored. Given L, R input secondary structures to the RNA Energy Barrier problem, the indices to this table are sets of the form $\{\ell, \ell_1 \dots \ell_p\}$, with $\ell \in L \cup \{(1, N)\}$ and $\ell_i \in L$, such that $\forall i\ \ell_i \subset \ell$, and $\forall i \neq j,\ \ell_i \parallel \ell_j$. The reason $(1, N)$ is a possible value for ℓ is that it defines an interval to which partial instances are restricted. Originally, there is no restriction and $\ell = (1, N)$. The partial instance associated to such a set is L', R' with $L' = L^{\ell}_{\text{IN}} \cap \left[\bigcap_{1 \leq i \leq p} L^{\ell_i}_{\text{OUT}}\right]$ and $R' = R^{\ell}_{\text{IN}} \cap \left[\bigcap_{1 \leq i \leq p} R^{\ell_i}_{\text{OUT}}\right]$. We also denote these structures by $L(\ell, \ell_1, \dots, \ell_p)$ and $R(\ell, \ell_1, \dots, \ell_p)$.

Informally, seeing L as a tree structure, the arcs $\{\ell, \ell_1, \dots, \ell_p\}$ define a "sub-tree" of L. ℓ sets the root of this sub-tree, while $\ell_1 \dots, \ell_p$ cut out some branches. Let us write

$$\mathcal{ST}(L) = \left\{(\ell, \ell_1, \dots, \ell_p) \in \left(L \cup \{(1, N)\}\right) \times L^p \mid \forall i\ \ell_i \subset \ell \text{ and } \forall i \neq j\ \ell_i \parallel \ell_j\right\}$$

for the set of all such "sub-trees", i.e. the set of all indices to the dynamic programming table.

Lemma 13. *Given an RNA structure L of arboricity Φ, $|\mathcal{ST}(L)| = O(\frac{n^{\Phi+1}}{\Phi!})$*

Proof. Let us start by noting that in an RNA structure, each arc is either terminal or contains a terminal arc nested in it. The set $\{\ell_i\}_{1 \leq i \leq p}$ being composed of arcs mutually not nested in one another, each of them contains (or is) a different terminal arc. As there are less than Φ terminal arcs in total, given ℓ, there are less than $\binom{n}{\Phi}$ possibilities for $\ell_1 \dots \ell_p$. Multiplied by the number of possibilities for ℓ, we get an upper bound of $(n+1) \cdot \binom{n}{\Phi} = O(\frac{n^{\Phi+1}}{\Phi!})$.

□

Theorem 21. *Algorithm 5 outputs a canonical (and therefore optimal) schedule in time $O(\frac{n^{\Phi+2}}{\Phi!})$.*

Proof. **Run-time** The initial call to SCHEDULE(L, R) corresponds to the entry $\ell = (1, N)$ and $\{\ell_1 \dots \ell_p\} = \emptyset$. Then, consider a call of SCHEDULE on two structures $L(\ell, \ell_1, \dots, \ell_p), R(\ell, \ell_1, \dots, \ell_p)$ respectively equal to $L^{\ell}_{\text{IN}} \cap \left[\bigcap_{1 \leq i \leq p} L^{\ell_i}_{\text{OUT}}\right]$ and $R^{\ell}_{\text{IN}} \cap \left[\bigcap_{1 \leq i \leq p} R^{\ell_i}_{\text{OUT}}\right]$, for $(\ell, \ell_1, \dots, \ell_p) \in \mathcal{ST}(L)$. The recursive calls to the "inside" and "outside" of some $\ell' \in L'$ (line 14-15) will give rise to the instances

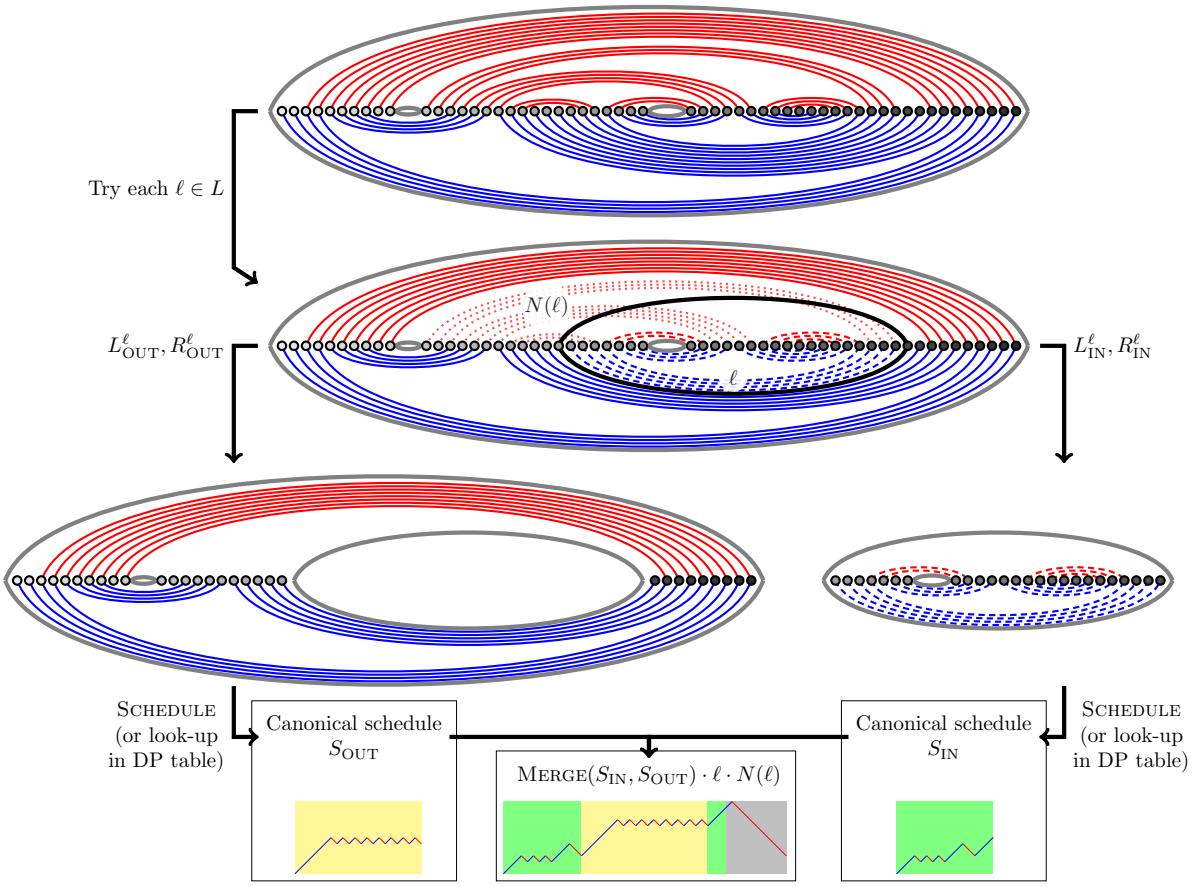


Figure 5.7: Illustration of Algorithm 5. Given a sub-instance L, R (bordered by gray ovals, top figure), each $\ell \in L$ is tried (middle figure), yielding two smaller sub-instances corresponding to the outside and inside of ℓ (bottom left and right figures). After solving each sub-instance independently, using the DP-table for memorization, a solution is obtained for (L, R) by merging both solutions and appending ℓ and all its neighborhood (which were not part of any sub-instance). Here the arboricity is 3 (there are 3 minimal arcs in L), so any border uses at most 4 arcs, giving the upper bound of $\binom{n}{4}$ on the number of sub-instances

corresponding to the elements of $\mathcal{ST}(L)$ $(\ell', \{\ell_i \mid \ell_i \subset \ell'\})$ (inside) and $(\ell, \ell', \{\ell_i \mid \ell_i \parallel \ell'\})$ (outside). By induction, all recursive calls to SCHEDULE are of these forms, and the indices to the memorization table are elements of $\mathcal{ST}(L)$. Conversely, any element $(\ell, \ell_1, \ldots, \ell_p)$ of $\mathcal{ST}$ sees its corresponding instance $L(\ell, \ell_1, \ldots, \ell_p), R(\ell, \ell_1, \ldots, \ell_p)$ emerge in some recursive call (e.g., taking the inside of ℓ in the first recursive call and then the outside of $\ell_1 \ldots \ell_p$). Their number is smaller than $O(\frac{n^{\Phi+2}}{\Phi!})$ by Lemma 13. Let now us call $c(L, R)$ the computational cost of SCHEDULE(L, R), and $i(L, R)$ the "internal" cost of SCHEDULE, i.e. of all lines of Algorithm 5 except lines 18-19 (recursive calls). Given $\ell \in L$, we write $L^{\ell}_{\text{IN}}, R^{\ell}_{\text{IN}}$ and $L^{\ell}_{\text{OUT}}, R^{\ell}_{\text{OUT}}$ the sub-instances composed of arcs strictly inside or outside of ℓ. Then, we have:

$$c(L, R) = i(L, R) + \sum_{\ell \in L} c(L^{\ell}_{IN}, R^{\ell}_{IN}) + c(L^{\ell}_{OUT}, R^{\ell}_{OUT})$$

Which, by induction, given the discussion above, allows to show that:

$$c(L, R) = \sum_{(\ell, \ell_1, \ldots, \ell_p) \in \mathcal{ST}(L)} i\left(L(\ell, \ell_1, \ldots, \ell_p), R(\ell, \ell_1, \ldots, \ell_p)\right)$$

$i(L, R)$ is $O(n^2)$ (linear MERGE for each $\ell \in L$), which yields $O(\frac{n^{\Phi+2}}{\Phi!})$ overall.

Correctness By Corollary 7, a canonical schedule S for G exists. Some element $\ell \in L$ is necessarily processed last, such that $S = S' \cdot \ell \cdot N(\ell)$. When ℓ is considered as part of the for loop line 17 of Algorithm 3, the candidate solution is $S'' \cdot \ell \cdot N(\ell)$, with $S'' =$ MERGE(S_{IN}, S_{OUT}). Sequence S'' is canonical by induction and the correctness of MERGE (Theorem 20). Thus, $S'' \preccurlyeq S'$ and the candidate solution is preferable to S, and therefore canonical.

□

5.7 Benchmarks

In this section, we report benchmark results for all of our algorithms. We first explain some details about the algorithm we implemented for directed pathwidth. Then, we present a general benchmark of Algorithm 4 and the directed pathwidth approach, on random (Erdös-Rényi) bipartite graphs. Last, we compare Algorithm 5 with the directed pathwidth approach on bipartite circle graphs, i.e. RNA instances.

Code availability. The code used for our benchmarks, including a Python/C++ implementation of our two algorithms, is available at https://gitlab.inria.fr/bmarchan/bisr-dpw (Algorithm 4 and directed pathwidth algorithm [211]) and https://gitlab.inria.fr/bmarchan/barrier-subtree (for Algorithm 5).

5.7.1 Implementation details

Directed pathwidth. We implemented and used an algorithm from Tamaki [189], with a runtime of $O(n^{\rho+2})$. This algorithm was originally published in 2011 [189]. In 2015, H.Tamaki and other authors

Algorithm 5 XP algorithm in Φ for Bipartite Independent Set Reconfiguration

Input: bipartite circle graph G (with sides L and R)
Output: a canonical schedule for G
Global variable: Dynamic programming table $M : (L, R) \to S$, storing input/output pairs for SCHEDULE.

function SCHEDULE(L, R):
  if (L, R) is in $M.keys()$ **then** ▷ If already computed then return;
    return $M[(L, R)]$;
  end if
  $M[(L, R)] = L \cdot R$ ▷ Initializing $M[(L, R)]$ with a simple value
  if $L = \emptyset$ **then**
    return $M[(L, R)]$
  end if
  if $\exists r \in R$ such that $N(r) = \emptyset$ **then**
    $M[(L, R)] = r \cdot$ SCHEDULE$(L, R \setminus \{r\})$
    return $M[(L, R)]$
  end if
  for ℓ `in` L **do**
    $S_{IN} =$ SCHEDULE(L_{IN}, R_{IN}) ▷ ℓ defines an inside and an outside
    $S_{OUT} =$ SCHEDULE(L_{OUT}, R_{OUT})
    $S' =$ MERGE$(S_{IN}, S_{OUT}) \cdot \ell \cdot N(\ell)$
    if $S' \preccurlyeq M[(L, R)]$ **then** ▷ If S' is preferable to $M[(L, R)]$
      $M[(L, R)] = S'$
    end if
  end for
  return $M[(L, R)]$
end function

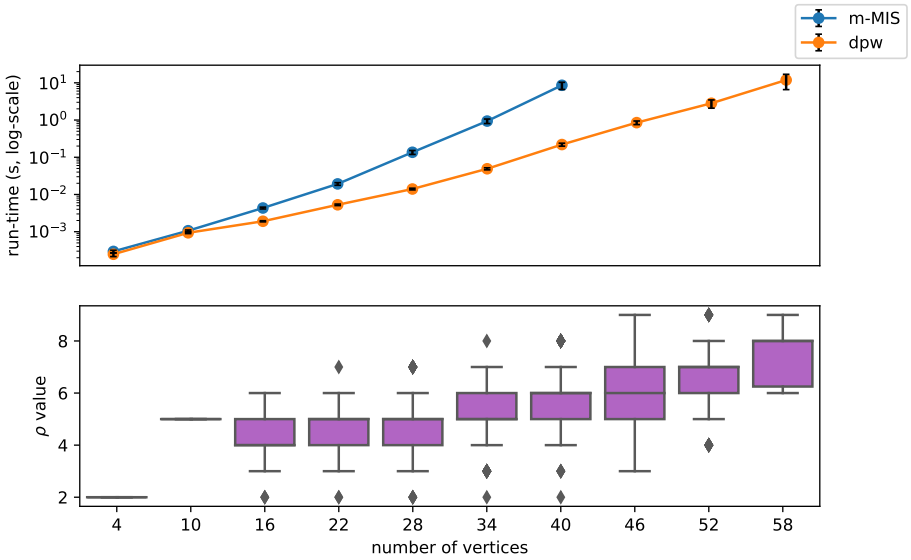


Figure 5.8: (top panel) Average run-time (seconds, log-scale) of our algorithms on random Erdös-Rényi bipartite graphs, with a probability of connection such that the average degree of a vertex is 5 (i.e $p = 5/n$). (bottom panel) Average parameter value of generated instances, as a function of input size.

described this algorithm as "flawed" in [201], and replaced it with another XP algorithm for directed pathwidth, with a run-time of $O(\frac{mn^{2\rho}}{(\rho-1)!})$.

Upon further analysis from our part, and discussions with H. Tamaki and the corresponding author of [201], it appears a small modification allowed to make the algorithm correct. In a nutshell, the algorithm involves *pruning* actions, and these need to be carried out *as soon as they are detected*. In [189], temporary solutions were accumulated before a general pruning step. With this modification, the analysis presented in [189] applies without modification, and yields a time complexity of $O(n^{\rho+2})$. The space complexity is unchanged at $O(n^{\rho+1})$. For completeness, a detailed re-derivation of the results of [189] is included in the full version of the article.

Mixed-MIS algorithm implementation. On Figure 5.8, the "m-MIS"-curve, corresponds to our mixed-MIS-based algorithm in $O(n^{2\rho}\sqrt{|V||E|})$. Compared to the algorithm presented in Algorithm 4, a more efficient rule is used in the non-separable case: we loop over all possible $r \in R$ and add $N(r) \cdot r$ to the schedule (instead of a single vertex $\ell \in L$).

5.7.2 Random bipartite graphs

Benchmark details. Figure 5.8 shows, as a function of the number of vertices, the average execution time of both our algorithms (top panel), as well as the distribution of parameter values (ρ - bottom panel), on a class of random bipartite graphs. These graphs are generated according to an

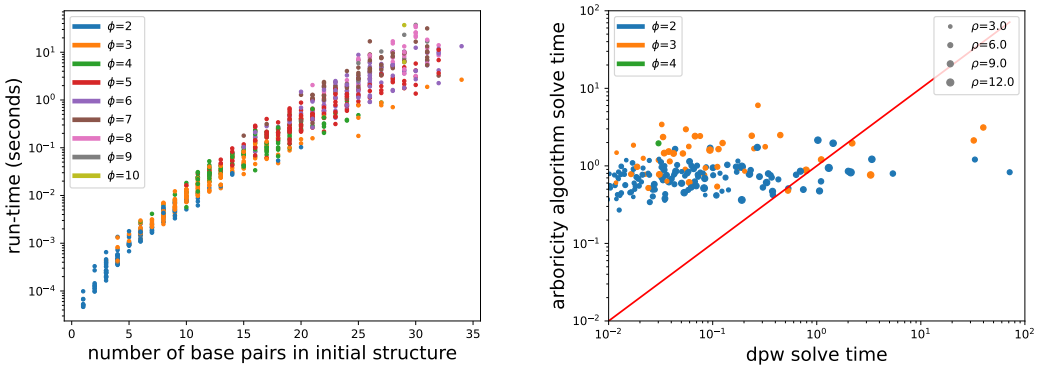


Figure 5.9: (left) Execution time of Algorithm 5 on pairs of random RNA secondary structures. Points are colored as a function of the smallest arboricity Φ between the two structures. As expected, exact computing the energy barrier between the structures tends to become more expensive for larger values of Φ. (right) Comparison with a scatter plot of the execution times of Algorithm 5 and our implementation of [211] (directed pathwidth algorithm), on random pairs of RNA secondary structures. The color of the points denote the arboricity (Φ) value while the size is the directed pathwidth (range ρ). Surprisingly, the execution time of the directed pathwidth algorithm, whose complexity is $O(n^{\rho+2})$, does not correlate with the value of ρ. It suggests the existence of a structural property of directed graphs emerging from RNA instances making [211] faster.

Erdös-Rényi distribution (each pair of vertices has a constant probability p of forming an edge). We use a connection probability of d/n, dependent on the number of vertices. It is such that the average degree of vertices is d. The data of our benchmark (Figure 5.8) has been generated with $d = 5$.

Comments on Figure 5.8. The difference in trend between the execution times of the two algorithms is quite coherent with the difference in their exponents ($n^{\rho+2}$ vs. $n^{2\rho+2.5}$).

5.7.3 random RNA instances (bipartite circle graphs)

Benchmark details. Figure 5.9 shows the average execution time of Algorithm 5 on random RNA instances, and compares it with the directed pathwidth algorithm (right panel). Random instances are generated according to the following model: two secondary structures L, R are chosen *uniformly* at random (within the space of all possible secondary structure). Base pairs are constrained to occur between nucleotides separated by a distance of at least $\theta = 1$ (left panel) and $\theta = 3$ (right panel).

Random secondary structure generation. The random RNA secondary structure of Figure 5.9 are obtained by *uniform sampling* of well-parenthesized strings of a given length N. Two parameters control the probability distributions: the minimal distance θ between an opening bracket and its corresponding closing bracket and the probability p_{pb} of being base-paired.

5.8 Conclusion

Motivated by the development of exact parameterized algorithms for the RNA Energy Barrier problem, we studied several parameterizations for Bipartite Independent Set Reconfiguration. For the *range* ρ of possible cardinalities for the independent sets along the reconfiguration as a parameter, we give a direct $O(n^2)$-space, $O(n^{2\rho+2.5})$-time algorithm (Algorithm 4), and a indirect $O(n^{\rho+1})$-space, $O(n^{\rho+2})$-time algorithm [211] through an equivalence with directed pathwidth.

In the case of RNA instances, i.e.BISR on bipartite circle graphs, we additionally study an *arboricity* parameter denoted Φ, that should intuitively be much smaller than the size of instances on natural RNA structures. For this parameter, we also provide an XP algorithm, with complexity $O(\frac{n^{\Phi}}{\Phi!})$ in space and $O(\frac{n^{\Phi+2}}{\Phi!})$ in time. This algorithm involves a novel *merge* procedure for optimal solutions of disjoint instances of *mimimum cumulative-cost* scheduling, which may be of independent interest.

The fixed-parameter tractability of Bipartite Independent Set Reconfiguration restricted to bipartite circle graphs, with respect to ρ, Φ and $\rho+\Phi$ remains open. It implies that the fixed-parameter tractability of directed pathwidth (i.e. ρ for BISR on general instances) also remains open. We nevertheless hope that this newly-drawn connection between a width parameter (directed pathwidth and a reconfiguration problem (BISR), may help shed new light onto this problem. In that respect, combining [215] and [200] to try to formulate an "obstacle theory" for directed pathwidth might be an interesting avenue.

Chapter 6

Conclusion

This PhD thesis has explored the possibility of applying parameterized algorithmics, especially graph width parameters, to structural RNA bioinformatics. In this conclusion, we give an account of the contributions, open problems, and research perspectives left to explore. This account is organized around two research axes. The first concerns treewidth, and the different ways it may be used to put *pseudoknots* into the picture. This axis encompasses Chapters 2, 3, and 4. The second axis is about the connections between directed graph parameters, scheduling problems, and kinetic aspects of RNA. It is connected to Chapter 5.

Axis 1: working with pseudoknots thanks to treewidth

Smooth parameterization of pseudoknots. Our analysis of treewidth values of RNA structures in the PDB database [138] in Chapter 1 (Figure 1.11, page 40) shows that the vast majority of pseudoknotted structures have reasonable treewidth ($\lesssim 10$). In other words, natural pseudoknotted RNA structures, taken as graphs, are not so far from being trees. It gives a smooth quantification of the complexity of RNA structures, starting at the base case treewidth$= 2$, which corresponds to conflict-free structures. Given the deep connections of treewidth with *dynamic programming* [73], an algorithmic design technique that has been used extensively in RNA bioinformatics [18, 24, 20], treewidth seems to be a very natural tool for the development of algorithms capable of handling pseudoknots.

Chapters 2, 3 and 4 are developments along this research direction. More precisely, Chapters 2 and 4 pave the way towards the development of efficient solutions for pseudoknotted *homolog search*, an open technological problem prior to this PhD thesis. As for Chapter 3, it adopts a slightly different viewpoint on treewidth. Indeed, it reinterprets tree decompositions as *dynamic programming schemes* for RNA FOLDING, or more generally as *multiple context-free grammars* for pseudoknotted RNAs.

The next two sub-sections describe these two themes in more details, starting with pseudoknotted

homolog search.

Towards pseudoknotted ncRNA search

This thesis contributes to making efficient pseudoknotted homolog search closer to reality in two ways. First, the results of Chapter 2 allow to compute *hierarchies* of structural models of increasing treewidth, with each level containing a *maximal* number of base-pairs given its treewidth. This hierarchy can be used for *hierarchical filtering* with existing treewidth-based methods for Structure-Sequence Alignment [62], the computational problem at the core of *homolog search*. Second, Chapter 4 gives a possible formulation for *pseudoknotted covariance model*. They can be seen as both a pseudoknotted generalization of `InfeRNAl`, and a generalization of the algorithm of [62] to include *stacking* terms and position-specific scores. The alignment complexity is parameterized by the treewidth of the consensus structure, and recovers the $O(n^3)$ complexity of `InfeRNAl` for treewidth$=2$. We now comment on the results of each chapter in a bit more detail, starting with Chapter 2.

Chapter 2 - Reducing treewidth to unlock FPT algorithms. Chapter 2 presents practical methods for reducing the treewidth of a graph while removing a minimum amount of edges. The core result is an FPT algorithm for Tree-Diet, the problem of reducing the width of a given tree decomposition while keeping as many edges as possible in the input graph. It was implemented (available at https://gitlab.inria.fr/amibio/tree-diet), and proofs-of-concepts of applications to RNA design and homolog search, were discussed in Chapter 2. Let us summarize the intuition behind the *hierarchical filtering* approach that it enables for homolog search. Let $\mathcal{A}$ be a set of base-pairs, and $\mathcal{A}'$ a reduced model of lower treewidth. In a nutshell, if aligning a *query* arc-annotated sequence $(Q, \mathcal{A}')$ with a target T yields an optimal score of $\text{OPT}(Q(\mathcal{A}'), T)$, then:

$$\text{OPT}(Q(\mathcal{A}), T) \leq \text{OPT}(Q(\mathcal{A}'), T) + \text{gain}(\mathcal{A}' \to \mathcal{A})$$

With $\text{gain}(\mathcal{A}' \to \mathcal{A})$ the best possible score contribution of the base-pairs in $\mathcal{A} \setminus \mathcal{A}'$. In other words, an upper bound for the quality of T as a candidate hit for the *full model* is obtained from a cheaper computation with a reduced model. In the case of homolog search, where a large set of targets T are tested as a "window" is slid on a sequence database, it allows to filter out some positions. In the equation above, the query model could also be a pseudoknotted covariance model (as described in Chapter 4).

Open problem - Graph Diet for RNA graphs. In Chapter 2, algorithms were developped for Tree-Diet, but graph simplifications preserving more edges could potentially be obtained by solving Graph-Diet, the direct problem of minimal edge removal to reduce the treewidth of an input graph G. For general graphs, we conjecture (Conjecture 1) that no practical algorithms can be obtained. But the question remains open for special classes of graphs. For instance, RNA structure graphs always have a $3D$ embedding in which each edge has a specific distance (depending on its chemical nature). They are also *Hamiltonian*, per the existence of a "backbone" for the molecule. As a starting point, we

saw in Chapter 2 that solving GRAPH DIET on a graph implies in particular being able to compute its treewidth (Theorem 6). The question of whether treewidth can be computed more efficiently on such a graph class compared to the general case would therefore be a first step.

Perspective - expanding the scope of TREE-DIET. To finish, note that the TREE-DIET algorithm can be applied to any tree decomposition and graph. This should be taken as motivation for developing more treewidth-based algorithms in Bioinformatics. Outside of bioinformatics, a notable example of treewidth application the computation of *marginals* in *constraint networks* [216]. The question of whether a TREE-DIET-like approach could be applied in such context is an interesting research direction. A related task, the *learning* of constraint networks of bounded treewidth is mentioned in the literature [217, 218].

Chapter 4 - Evaluating pseudoknotted alignment methods, and designing new ones. Chapter 4 first presents benchmark results for `LiCoRNA` (an implementation of [62], available at `https://licorna.lri.fr/`). The purpose was to evaluate the capacity of `LiCoRNA`, a pseudoknot-aware STRUCTURE-SEQUENCE ALIGNMENT method, to produce good quality seed alignments for pseudoknotted `Rfam` families. The contrasted results hint at the possibility that `Rfam`, based on `InfeRNAl`, aligns pseudoknots based mostly on *sequence conservation*. These difficulties could be overcome by formulating a fully-featured pseudoknotted generalization of `InfeRNAl`, which we develop in a second part of the chapter. We call this generalization *pseudoknotted covariance models*. They are based on *multiple context-free grammars* as a generalization of the context-free grammars underlying `InfeRNAl`, and a treewidth-based `LiCoRNA`-style algorithm for aligning sequences onto the model.

Perspective - implementation and evaluation of PKCMs. A clear research perspective is the implementation and benchmark of pseudoknotted covariance models. A critical point is to achieve good practical performance for the alignment of an input sequence onto a model. Indeed, the treewidth-based algorithm for doing so runs in $O(f(tw) \cdot n^{tw+1})$, with f an exponential function and tw the treewidth of the consensus structure. The base case is $O(n^3)$, which corresponds to the complexity of this problem for covariance models. Such a complexity was actually considered *heavy* in the early development of covariance models, and `InfeRNAl` only became competitive after a series of optimizations [180, 61]. Similar optimizations will therefore have to be applied to our case as well, if we want to reach competitive run-times. The speed-up of individual alignment instances with a *tree diet* approach (Chapter 2), as outlined at the end of Chapter 4, could be part of the answer. Then, a comprehensive benchmark experiment should include the possibility of fetching new hits for each pseudoknotted ncRNA family, and assess whether they form a more coherent set compared to the current composition of the family.

Pseudoknotted grammar generation

Chapter 3 - Automated DP scheme generation for RNA FOLDING. This thesis also explored the possibility of using tree decompositions as a *grammar generation tool*. In Chapter 3, we developed this

way a framework that takes as input a pseudoknotted pattern, and outputs a set of dynamic programming equations that solve the RNA FOLDING problem restricted to this pattern. The pattern is specified as a conflicted set of *bands*, also called *fatgraph*. Each band in this fatgraph is to be thought of as an *helix*, i.e. a succession of well-nested base-pairs, of arbitrary size. The key result is that a tree decomposition of a *minimal representative expansion* of a fatgraph, where each helix has length 5, can be put into a *canonical form*, generalizable to any tree decomposition in the family. A *dynamic programming scheme*, i.e. the definitions of dynamic programming tables and recursive relations connecting them, can then be inferred from the structure of the canonical tree decomposition. Importantly, the time complexity of the generated scheme is linked to the *width of the tree decomposition*. The complexity of famous hand-designed DP schemes [29, 16] is then recovered automatically.

Perspective - Automated code generation. Chapter 3 presented some prototype automated code generation from the decomposition of a fatgraph. Examples of missing features include *stacking* energy terms and the possibility of *ensemble* applications, i.e. sampling in the Boltzmann distribution. To implement a full efficient pipeline for automated code generation with these features is a clear direction for future work.

Open problem - Optimality. Even though the DP schemes we generate match the complexities of hand-written schemes, it would be nice to prove some form of general *optimality*. A first step would be *space optimality*, i.e. that the DP schemes we output use as few indices as possible. A rough idea for proving such a result would be to show that any DP scheme covering the language of structures specified by a fatgraph could be used to get a *tree decomposition* for these structures, and in particular for the *minimal representative expansion* we use in our framework. The intuition is then that the width of that decomposition would correspond to the largest number of indices of any of the DP table, and therefore could not be smaller than the treewidth. Generating a DP scheme with our method, starting with an optimal tree decomposition, would reach that lower bound and therefore be space-optimal.

Perspective - Connection with graph grammars. One of the characterizations of treewidth is through *hyperedge-replacement grammars* [219]. In short, a *hyper-edge replacement* grammar (HRG) is a graph grammar, and graphs of treewidth $\leq k$ are exactly the graphs obtained by HRGs when bounding the "cardinality" of a rule by k [220, 221]. Such a connection raises the question of whether a grammar of *RNA structures of bounded treewidth* could be formulated. This could be seen as generalizing the framework of Chapter 3, where such a grammar would be able to explore *all fatgraphs* of a treewidth smaller than a threshold. Potential applications could be an algorithm for RNA FOLDING that explores all structures below a certain treewidth value.

Axis 2 - directed width parameters, reconfiguration and RNA kinetics

This PhD thesis also explored the possibility of applying parameterized algorithmics for the *exact* computation of RNA energy barriers. This time, the width parameter it is connected to is *directed*

pathwidth, a width measure for directed graphs. This work also highlighted interesting connections with a class of scheduling problems (*minimum cumulative-cost scheduling*) and *independent set reconfiguration*. A lot of parameterized complexity questions remain open, calling for future work.

Chapter 5 - Parameterized algorithmics for independent set reconfiguration and RNA kinetics. The RNA Energy Barrier problem asks, given two conflict-free secondary structures L and R, how to transform L into R with a series of arc addition and removals. The constraints are that at any point along the reconfiguration, the arcs must be crossing-free, and that the number of arcs should be maximized at all times. By taking the *conflict-graph* of arcs in $L \cup R$, we get a bipartite graph G in which a valid structure is an *independent set*. The RNA Energy Barrier problem is then a *bipartite independent set reconfiguration* on G.

We formulate in Chapter 5 two XP algorithms parameterized by ρ, the maximum allowed distance to $\alpha(G)$, the MIS size of G. One of them stems from a connection with *directed pathwidth*, which measures how close a directed graph is from being a *DAG*[1]. More precisely, the energy barrier between L and R is the directed pathwidth of an *auxiliary graph*, obtained from a maximum matching of G.

We also define and look at an *arboricity* parameter, denoted Φ, and equal to the minimum number of hairpin loops in L or R. For that parameter we also find an XP algorithm. It involved solving the non-trivial problem of *merging* optimal solutions for two disconnected instances into one global optimal solution. This problem had also been set in a context of *minimum cumulative-cost scheduling* [207], where our solution could potentially find other applications.

Open Problem - directed pathwidth. Our first open problem is whether RNA Energy Barrier is FPT by ρ or not. Interestingly, the RNA Energy Barrier problem parameterized by ρ comes down exactly to the question "Is $\text{dpw}(H) \leq \rho$?" for H an auxiliary graph built from a maximum matching of a *bipartite circle graph* (the conflict graph G). Whether "Is $\text{dpw}(H) \leq \rho$?" is FPT by ρ or not for *general graphs* is an important open problem in directed structural graph theory [197, 211]. Answering this open question in the positive would therefore yield an FPT sub-case of directed pathwidth, which would be interesting in itself. On the contrary, proving $W[t]$-hardness (for some t) would imply the same for directed pathwidth, and would be a major result.

Open problem - parameterization by Φ. The questions of whether RNA Energy Barrier is FPT when parameterized by Φ, and by the combination $\Phi + \rho$, are also open. From an RNA structure point of view, given the tendency of natural structures to form *stacks* (stems), Φ is particularly relevant.

Perspective - more realistic energy models. One shortcoming of our approach is the restriction to a simple "number of base-pairs" energy model. Although generalization of all our algorithms to

[1] Directed Acyclic Graph

the weighted case is possible, the inclusion of *stacking* would get us significantly closer to realism. One first step in that research direction would be ignore isolated base-pairs, and treat each potential stacking as one unit.

Final word

In a context of increased use of Machine Learning approaches in Bioinformatics in general, this thesis arguably shows that exploring exact solutions for textbook problems is still relevant. It is a motivation for keeping in touch with *recent developments in algorithmic research*, to see if they can shed new light on well-studied problems. Parameterized algorithmics is particularly promising in that respect, with *width parameters* still being a matter of current research [222]. As an example of recent important development, the 2017 PACE challenge (https://pacechallenge.org/), a treewidth solver competition, gave rise to the algorithm of [82], which was used extensively to compute tree decompositions in this PhD work. On an another topic, one can also not exclude the development in the upcoming years of algorithmic techniques for *directed pathwidh*, which would have direct applications in RNA kinetics. Taking the opposite view, bringing concrete hard Bioinformatics problems to theoreticians can help motivate targeted theoretical developments. For instance, whereas there is no clear consensus as to what width measure is best for *directed graphs*, the connection we drew between directed pathwidth and RNA kinetics may help single out one with more applicative potential, making it interesting in itself.

Appendices

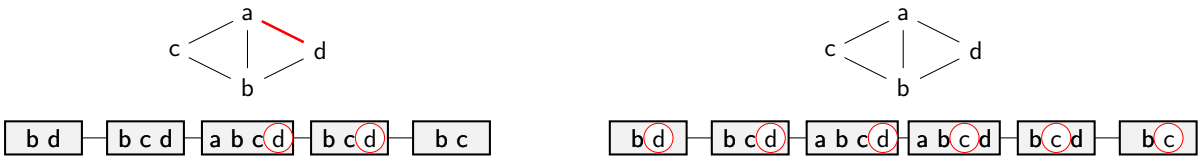


Figure 1: Left: A graph and a path-decomposition whose optimal 1-tree-diet loses an edge (ad). However, duplicating the bag abcd (right) yields a tree-decomposition with a lossless 1-tree-diet.

A Appendices to Chapter 2

A.1 Editing Trees before the Diet

Any tree decomposition can be transformed into a binary one through the duplications of bags having more than 2 children. To do so in practice, one will, as long as the tree decomposition is not binary, apply the following transformation:

1. Find a bag X with children $Y_1, \ldots, Y_\Delta$ and $\Delta > 2$.
2. Introduce a new bag X' with the same content as X and locally modify the tree decomposition in the following way: X will now have Y_1 and X' as children, while X' will have $Y_2 \cdots Y_\Delta$.

When it is no longer possible to apply this transformation, the tree decomposition is binary. For each bag having originally $\Delta > 2$ children in the decomposition, $\Delta - 1$ new bags have been introduced. In total, with N_{bags} the original number of bags in the decomposition, strictly less than N_{bags} new bags have been introduced (each new bag is associated to an edge of the original tree decomposition).

This transformation is in fact the first step towards obtaining a *nice tree decomposition* [75, 71].

A question that arises then is what impact these modifications may have on the output of Tree-Diet, when applied to the tree decomposition given as input. We argue that duplication operations (as used above to get a binary tree decomposition) can only improve the solution, i.e decrease the number of *lost* edges. Indeed, within the coloring formulation of the problem, new bags yield new opportunities for an edge to be *represented*, with both its end-points green in some bag. See Figure 1 for an illustration.

More generally, any operation on the input tree decomposition that does not suppress any of the original bags can only improve the solution to the Tree Diet problem. We do not tackle here the problem of finding the best edition operations to apply onto a tree decomposition given as input to Tree Diet, which is an a priori difficult task.

A.2 Pseudo-code

Algorithm 6 and 7 present a pseudo-code of our dynamic programming algorithm for Tree Diet, with a memoization approach. The C++/pybind11 [137] implementation is available at https://gitlab.inria.fr/amibio/tree-diet.

Note that the implementation allows to solve a more general *weighted* version of Tree Diet, where each edge is given a weight, and the objective is to find a $(tw - tw')$-diet of the input tree decomposition preserving a set of edges of maximum total weight.

In the context of RNA applications, this feature allows to favour as much as possible preservation of the backbone of RNA molecules, i.e. edges between consecutive nucleotides along the string, by assigning them a weight greater than the number of non-backbone edges.

Edge weights are passed to the function in the form of a dictionary/map W associating a real weight to each edge. Within Algorithm 6, the only place where it is taken into account is the the $count$ function, which computes the weight of edges accounted for by the bag that is currently visited.

A.3 Correctness of the rejection-based sampling of RNA designs

A recent method for RNA design, called RNAPond [48], implements a sampling approach to tackle the inverse folding of RNA. Targeting a secondary structure S of length n, it performs a Boltzmann-weighted sampling of sequences and, at each iteration, identifies Disruptive Base Pairs (DBPs) that are not in S, yet are recurrent in the Boltzmann ensemble of generated sequences. Those base pairs are then added to a set $\mathcal{D}$ of DBPs, and excluded in subsequent generations through an assignment of non-binding pairs of nucleotides, outside of $\mathcal{B} := \{(\mathtt{G},\mathtt{C}), (\mathtt{C},\mathtt{G}), (\mathtt{A},\mathtt{U}), (\mathtt{U},\mathtt{A}), (\mathtt{G},\mathtt{U}), (\mathtt{U},\mathtt{G})\}$.

At the core of the method, one finds a random generation algorithm which takes as input a secondary structure S and a set $\mathcal{D}$ of DBPs. The algorithm generates from the set $\mathcal{W}_{S,\mathcal{D}}$ of sequences $w \in \{\mathtt{A},\mathtt{C},\mathtt{G},\mathtt{U}\}^n$ which are: i) compatible with all $(i,j) \in S$, *i.e.* $(w_i, w_j) \in \mathcal{B}$; and ii) incompatible with all $(k,l) \in \mathcal{D}$, *i.e.* $(w_k, w_l) \notin \mathcal{B}$. The algorithm then enforces a (dual) Boltzmann distribution over the sequences in $\mathcal{W}_{S,\mathcal{D}}$:

$$\forall w \in \mathcal{W}_{S,\mathcal{D}} : \mathbb{P}(w \mid \mathcal{D}, S) = \frac{e^{-\beta.E_{w,S}}}{\mathcal{Z}_{S,\mathcal{D}}} \quad \text{with} \quad \mathcal{Z}_{S,\mathcal{D}} := \sum_{w' \in \mathcal{W}_{S,\mathcal{D}}} e^{-\beta.E_{w',S}} \tag{1}$$

where $\beta > 0$ is an arbitrary constant akin to a temperature. Yao *et al.* describe an algorithm which generates k sequences in $\Theta(k(n+|\mathcal{D}|))$ time, after a preprocessing in $\Theta(n.|\mathcal{D}|.4^{tw})$ time and $\Theta(n.4^{tw})$ space, where tw is the treewidth of the graph having edges in $S \cup \mathcal{D}$.

The discrepancy in the preprocessing and sampling complexities suggests an alternative strategy, utilizing rejection on top of a relaxed sampling. Namely, we consider a rejection algorithm, which starts from a relaxation $(S', \mathcal{D}')$ of the initial constraints ($S' \cup \mathcal{D}' \subset S \cup \mathcal{D}$), and iterates Yao *et al.*'s algorithm to generate sequences in $\mathcal{W}_{S',\mathcal{D}'} \supset \mathcal{W}_{S,\mathcal{D}}$, rejecting those outside of $\mathcal{W}_{S,\mathcal{D}}$, until k suitable ones are obtained. The rejection algorithm generates a given sequence $w \in \mathcal{W}_{S,\mathcal{D}}$ on its first attempt with probability $p := e^{-\beta.E_{w,S}}/\mathcal{Z}_{S',\mathcal{D}'}$ and, more generally, after r rejections with probability $(1-q)^r\, p$ with $q := \mathcal{Z}_{S,\mathcal{D}}/\mathcal{Z}_{S',\mathcal{D}'}$. The overall probability of emitting w is thus

$$p \cdot \sum_{r \geq 0} (1-q)^r = \frac{p}{q} = \frac{e^{-\beta.E_{w,S}}}{\mathcal{Z}_{S,\mathcal{D}}} = \mathbb{P}(w \mid \mathcal{D}, S).$$

Algorithm 6 Dynamic programming algorithm for TREE-DIET.

Input: Tree-decomposition $\mathcal{T}$, graph G, target width tw', edge weights W
Output: Maximum total weight of a set of realizable/non-lost edges in a $(tw - tw')$-diet of $\mathcal{T}$
Side product: A filled table $c[X_i, f]$, $\forall X_i$ bag and f coloring of X_i

```
1: function optim_num_real_edges(X_i, f, G, tw', W):
2:     if c[X_i, f] already computed then
3:         return c[X_i, f];
4:     end if
5:     if |f^{-1}(o) ∪ f^{-1}(r)| ≤ (|X_i| − tw' − 1) then
6:                                                  ▷ //not enough removals
7:         c[X_i, f] = −∞
8:         return c[X_i, f]
9:     end if
10:    if X_i == leaf then
11:        c[X_i, f] = 0
12:        return c[X_i, f]
13:    end if
14:    int ans = −∞;
15:    for m ∈ orange_maps(X_i, f) do
16:        int ans_m = 0;
17:        for Y_j ∈ X_i.children do int ans_j = −∞;
18:            for f'_j ∈ compatible(f, m, X_i, Y_j) do int val = 0;
19:                val += count(f, f'_j, W);
20:                val += optim_num_real_edgesY_j, f'_j, G, tw';
21:                if val ≥ ans_j then ans_j = val;
22:                end if
23:            end for
24:            ans_m += ans_j
25:        end for
26:        if ans_m ≥ ans then
27:            ans = ans_m;
28:        end if
29:    end for
30:    c[X_i, f] = ans return c[X_i, f]
31: end function
```

Algorithm 7 Backtracking procedure for TREE-DIET.

Input: Tree-decomposition $\mathcal{T}$, graph G, target width tw', table c, edge weights W
Output: Optimal $(tw - tw')$-diet-valid coloration for $\mathcal{T}$

```
1: function OPTIMAL_COLORING(X_i, f, G, tw', c):
2:     if X_i == leaf then return ∅;
3:     end if
4:     coloring C = ∅;
5:     for m ∈ orange_maps(X_i, f) do
6:         int ans_m = 0;
7:         coloring best_fjs = [];
8:         for Y_j ∈ X_i.children do
9:             int best_valj = −∞;
10:            int best_fj = ∅;
11:            for f'_j ∈ compatible(f, m, X_i, Y_j) do
12:                int val = 0;
13:                val += count(f, f'_j, W);
14:                val += c[Y_j, f'_j];
15:                if val ≥ best_valj then
16:                    best_valj = val;
17:                    best_fj = f'_j;
18:                end if
19:            end for
20:            ans_m += best_valj;
21:            best_fjs.add(best_fj);
22:            if ans_m == c[X_i, f] then
23:                C+ = [f'_j for f'_j in best_fjs];
24:                C+ = [optimal_coloringY_j, f'_j, G, tw', c for f'_j in best_fjs];
25:                break                                   ▷ // break loop over m
26:            end if
27:        end for
28:    end for
29: end function
```

In other words, our relaxed generator coupled with the rejection step, represents an unbiased algorithm for the Boltzmann distribution of Eq. (1) over $\mathcal{W}_{S,\mathcal{D}}$.

Meanwhile, the average-case complexity can be impacted by the strategy. Indeed, the relaxed instance $(S', \mathcal{D}')$ can accelerate the preprocessing due to a reduced treewidth $tw' \leq tw$. The rejection step only increases the expected number of generations by a factor $\overline{q} := \mathcal{Z}_{S',\mathcal{D}'}/\mathcal{Z}_{S,\mathcal{D}}$, representing the inflation of the sequence space, induced by the relaxation of the constraints. Overall, the average-case time complexity of the rejection algorithm is in $\Theta(n \cdot |\mathcal{D}'| \cdot 4^{tw'} + k \cdot \overline{q} \cdot (n + |\mathcal{D}'|))$ time and $\Theta(n \cdot 4^{tw'})$ space. This space improvement is notable when $tw' < tw$, and could be key for the practical applicability of the method, especially given that memory represents the bottleneck of most treewidth-based DP algorithms.

A.4 Lower bound for the min. alignment cost from simplified models

Here, we justify the filtering strategy described in Section 2.5.2. Namely, we formally prove that, given a structured RNA S and a targeted genomic region w, a lower bound for the minimal alignment cost of S and w can be obtained from the minimal alignment cost of some $S' \subseteq S$ and w. If this lower bound for $S' \subseteq S$ is higher than the specified cutoff ε, then there is no need to align w to the full model S, as the resulting cost is guaranteed to stay above the selection cutoff ε.

Let S be an arc-annotated sequence of length m (S_i denotes the ith character of S), w be a target (flat) sequence of length m, and $\mu : [1, n] \to [1, m] \cup \{\perp\}$ represents an alignment[2]. We consider the following cost function, adapted from eciteRinaudo2012, which quantifies the quality of an alignment μ for S and w:

$$C(S, w, \mu) = \sum_{\substack{i \text{ unpaired in } S, \\ k:=\mu_i}} \gamma(S_i, w_k) + \sum_{\substack{(i,j)\in S, \\ (k,l):=(\mu_i,\mu_j)}} \phi(S_i, S_j, w_k, w_l) + \sum_{g \in gaps(S)} \lambda_q(g) + \sum_{g \in gaps(w)} \lambda_T(g)$$

where

- $\gamma(a, b)$ returns the *substitution cost* which penalizes (mismatches) or rewards (matches) the substitution of a into b (set to 0 and handled in gaps if $b = \perp$);
- $\phi(a, b, c, d)$ returns a *base pair substitution cost*, penalizing (arc breaking) or rewarding (conservation or compensatory mutations) the transformation of nucleotides (a, b) into nucleotides/gaps (c, d) (set to 0 and handled in gaps if $(c, d) = (\perp, \perp)$);
- λ_S and λ_T penalize gaps introduced by μ respectively in S and w (affine cost model).

[2] An alignment μ is subject to further constraints, notably including some restricted form of monotonicity, when represented as a function. However, those constraints are reasonably intuitive and we omit them in this discussion for the sake of simplicity.

Given this definition, consider a simplified model $S' \subset S$, associated with a minimal cost

$$c' := \min_{\mu} C(S, w, \mu)$$

and denote by $c^\star$ the minimal cost of the full model S, we have the following inequality.

Proposition 13.

$$c' - \sum_{\substack{i \text{ unpaired in } S,' \\ \text{paired in } S}} \max_{b} \gamma(S_i, b) + \sum_{(i,j)\in S\setminus S'} \min_{a,b} \phi(S_i, S_j, a, b) \leq c^\star \tag{2}$$

Proof. For any alignment, we have, per the definition of $C(S, w, \mu)$:

$$C(S, w, \mu) = C(S', w, \mu) - \sum_{\substack{i \text{ unpaired in } S,' \\ \text{paired in } S, \\ \text{and } k:=\mu_i}} \gamma(S_i, w_k) + \sum_{\substack{(i,j)\in S\setminus S' \\ \text{s.t. } (k,l):=(\mu_i,\mu_j)}} \phi(S_i, S_j, w_k, w_l).$$

Minimizing over all alignment μ, one obtains

$$\min_{\mu} C(S, w, \mu) = \min_{\mu} C(S', w, \mu) - \sum_{\substack{i \text{ unpaired in } S,' \\ \text{paired in } S, \\ \text{and } k:=\mu_i}} \gamma(S_i, w_k) + \sum_{\substack{(i,j)\in S\setminus S' \\ \text{s.t. } (k,l):=(\mu_i,\mu_j)}} \phi(S_i, S_j, w_k, w_l).$$

Independently minimizing each term of the right-hand-side, we obtain a first lower bound

$$c^\star \geq c' - \max_{\mu} \sum_{\substack{i \text{ unpaired in } S,' \\ \text{paired in } S, \\ \text{and } k:=\mu_i}} \gamma(S_i, w_k) + \min_{\mu} \sum_{\substack{(i,j)\in S\setminus S' \\ \text{s.t. } (k,l):=(\mu_i,\mu_j)}} \phi(S_i, S_j, w_k, w_l).$$

further coarsened by an independent optimization of the elements in the sums

$$\begin{aligned} c^\star &\geq c' - \sum_{\substack{i \text{ unpaired in } S,' \\ \text{paired in } S}} \max_{\mu} \gamma(S_i, w_k) + \sum_{(i,j)\in S\setminus S'} \min_{\mu} \phi(S_i, S_j, w_k, w_l) \\ &= c' - \sum_{\substack{i \text{ unpaired in } S,' \\ \text{paired in } S}} \max_{a} \gamma(S_i, a) + \sum_{(i,j)\in S\setminus S'} \min_{a,b} \phi(S_i, S_j, a, b). \end{aligned}$$

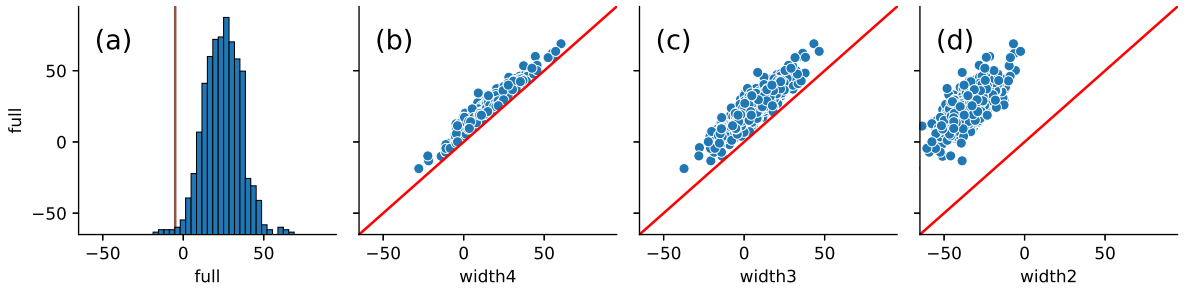


Figure 2: (a) Histogram of alignment scores obtained by aligning the full structure ($tw = 5$) model of the Twister ribozyme (pdb-id: 4OJI) with $\kappa \cdot n$-sized windows in a 10kb region of the 5^{th} chromosome of *S. bicolor*. A vertical line is positioned at the ϵ threshold. (b;c;d) Corrected alignment scores obtained for reduced-treewidth models for each window, plotted against the corresponding score of the full model. The corrected alignment score indeed acts as a lower bound to the full-model score (points above the $y = x$ red line), allowing an iterative filtering strategy.

where the last line is obtained by considering the worst-case contributors to nucleotides and base pairs substitutions. Importantly, the right-hand side no longer depends on μ any more, and can be used to easily computed a corrected score/lower bound. □

The corrected expression, shown in the left hand side of Equation (2) allows, when lower than a cutoff ε, to safely discard w as a potential hit for the full model S. This corrected score is plotted in Figure 2.10A, allowing for a gradual reduction of the search space for ε-admissible hits. We show in Figure 2 the corrected scores obtained for simplified structures S' of various treewidths, plotted against the scores of the full target structure.

B Appendix to Chapter 3

This Appendix to Chapter 3 contains detailed examples of dynamic programming scheme generation, on Figure 3.

C Appendix to Chapter 4

This Appendix to Chapter 4 contains full benchmark data for `LiCoRNA`, on Figures 4, 5, 6, 7, 8 and 9.

D Appendices to Chapter 5

D.1 Directed pathwidth definition

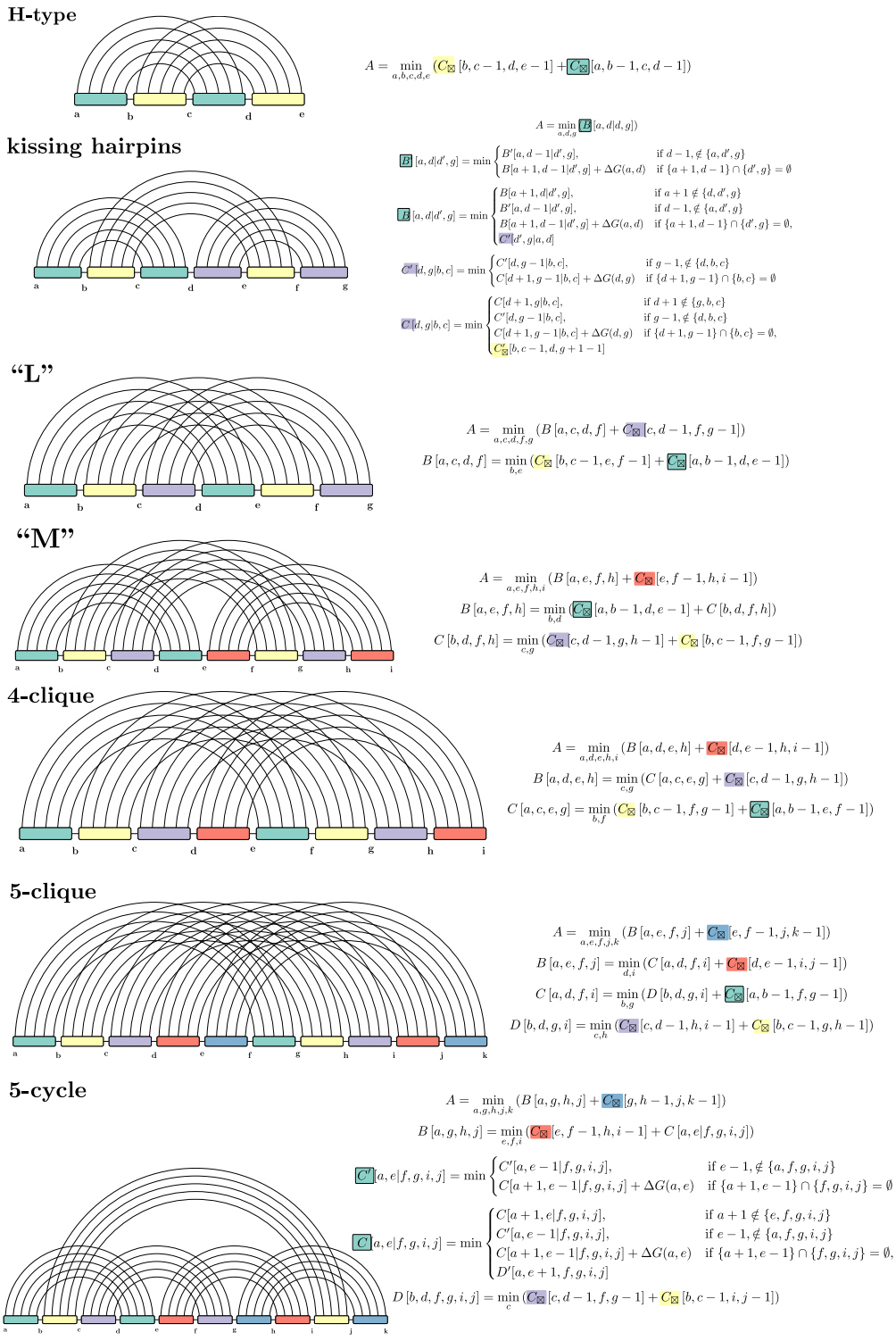


Figure 3: Minimal representative expansions and final equations for the examples of Table 3.1. The equations have been automatically generated, and the pipeline code is freely available at https://gitlab.inria.fr/bmarchan/auto-dp. In particular, the optimal tree decompositions were computed using an exact algorithm proposed by Tamaki [82].

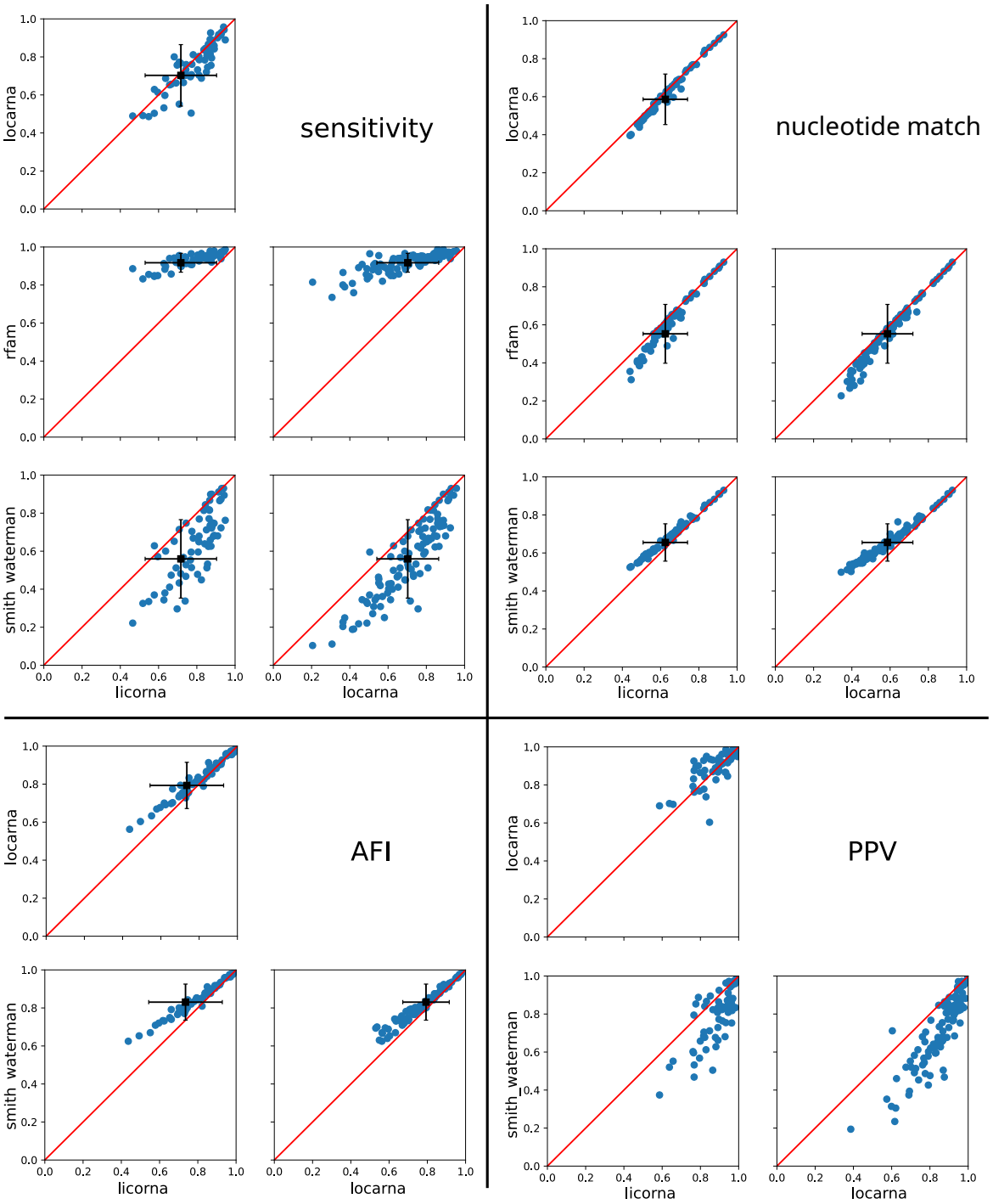


Figure 4: Comparison of AFI, sensitivity, PPV and nucleotide match values between the different methods, for **Experiment 1**, i.e. the alignment of pairs of seed sequences from pseudoknotted `Rfam` families. Each point in the scatter plot is one pseudoknotted `Rfam` family. The overall picture is contrasted, but a rough hierarchy of the tools may be given for each metric. For AFI, `Smith-Waterman`$\gtrsim$ `LocARNA`$>$ `LiCoRNA`. In terms of PPV, `LiCoRNA`$\simeq$ `LocARNA`$>$ `Smith-Waterman`. For both AFI and PPV, comparing to the `Rfam` alignment is pointless as all values would be equal to 1 for the seed alignment. For sensitivity, `Rfam`$>$ `LiCoRNA`$\simeq$ `LocARNA`$>$ `Smith-Waterman`. Finally, in terms of nucleotide match (sequence conservation) `Smith-Waterman`$>$ `LocARNA` $\simeq$ `LiCoRNA` $>$ `Rfam`. More interpretation of these results is given in the main text.

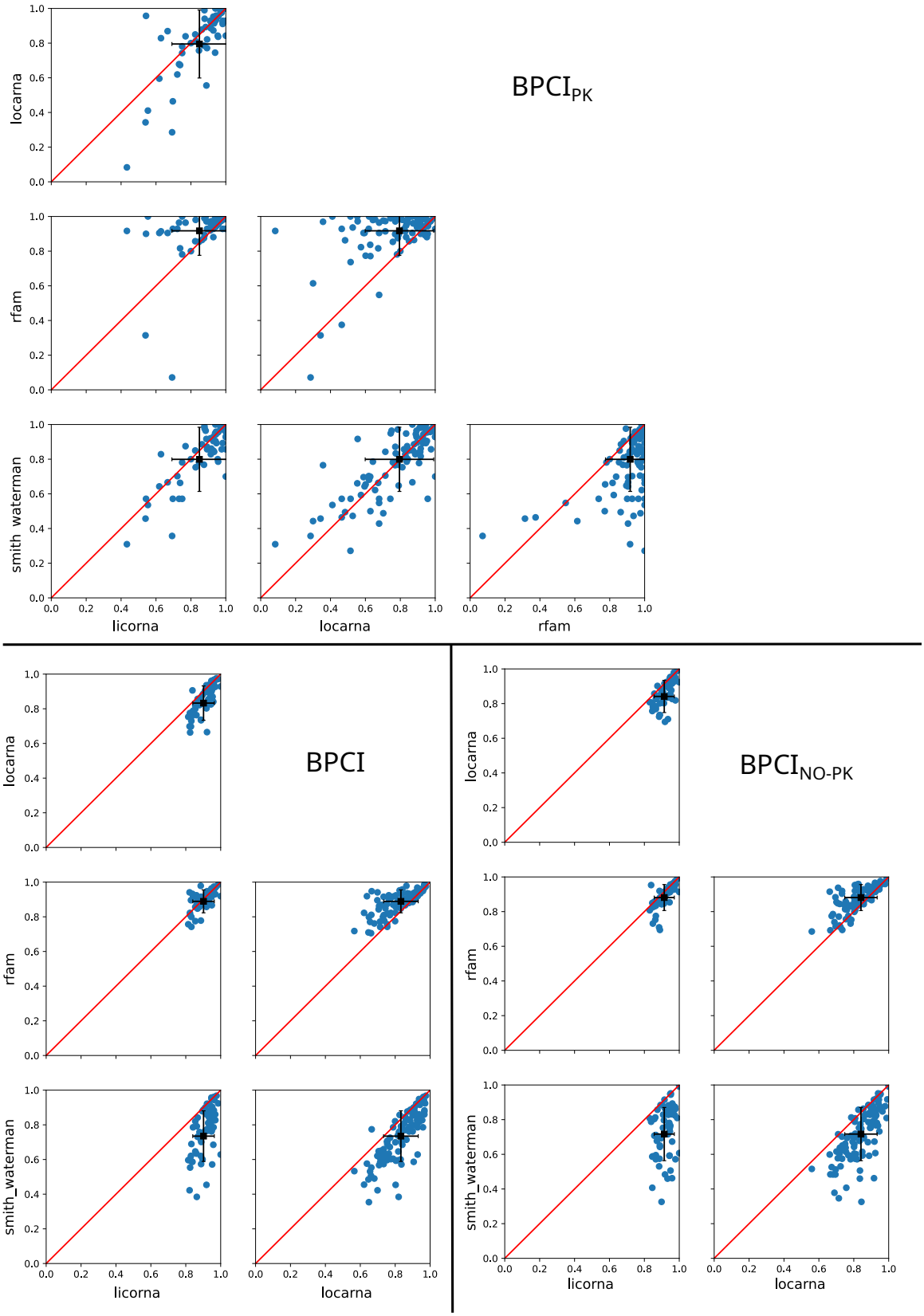


Figure 5: Comparison of BPCI values obtained for **Experiment 1** with `LiCoRNA`, `Rfam`,`LocARNA` and `Smith-Waterman`.

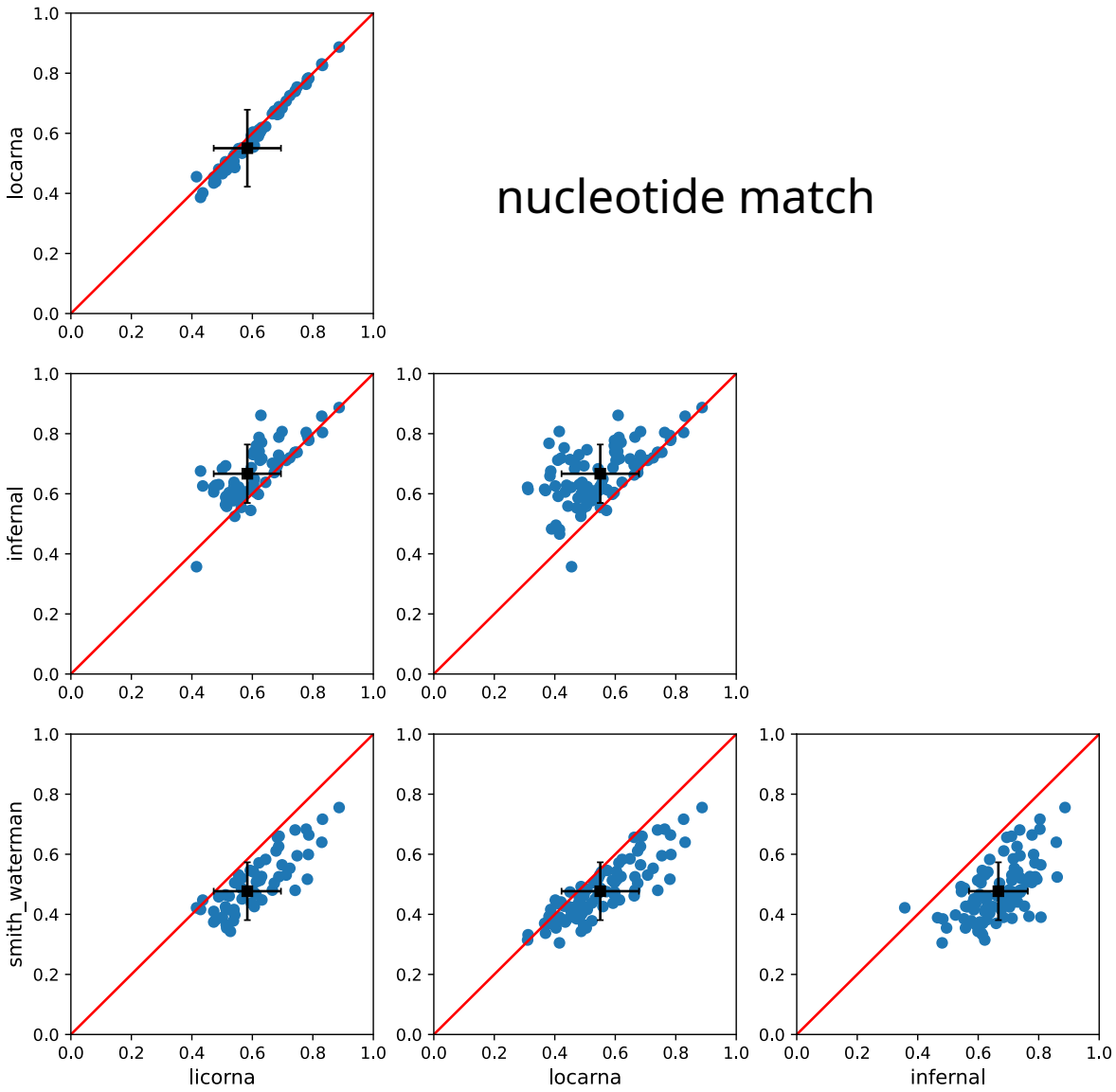


Figure 6: Nucleotide match results for **Experiment 2**.

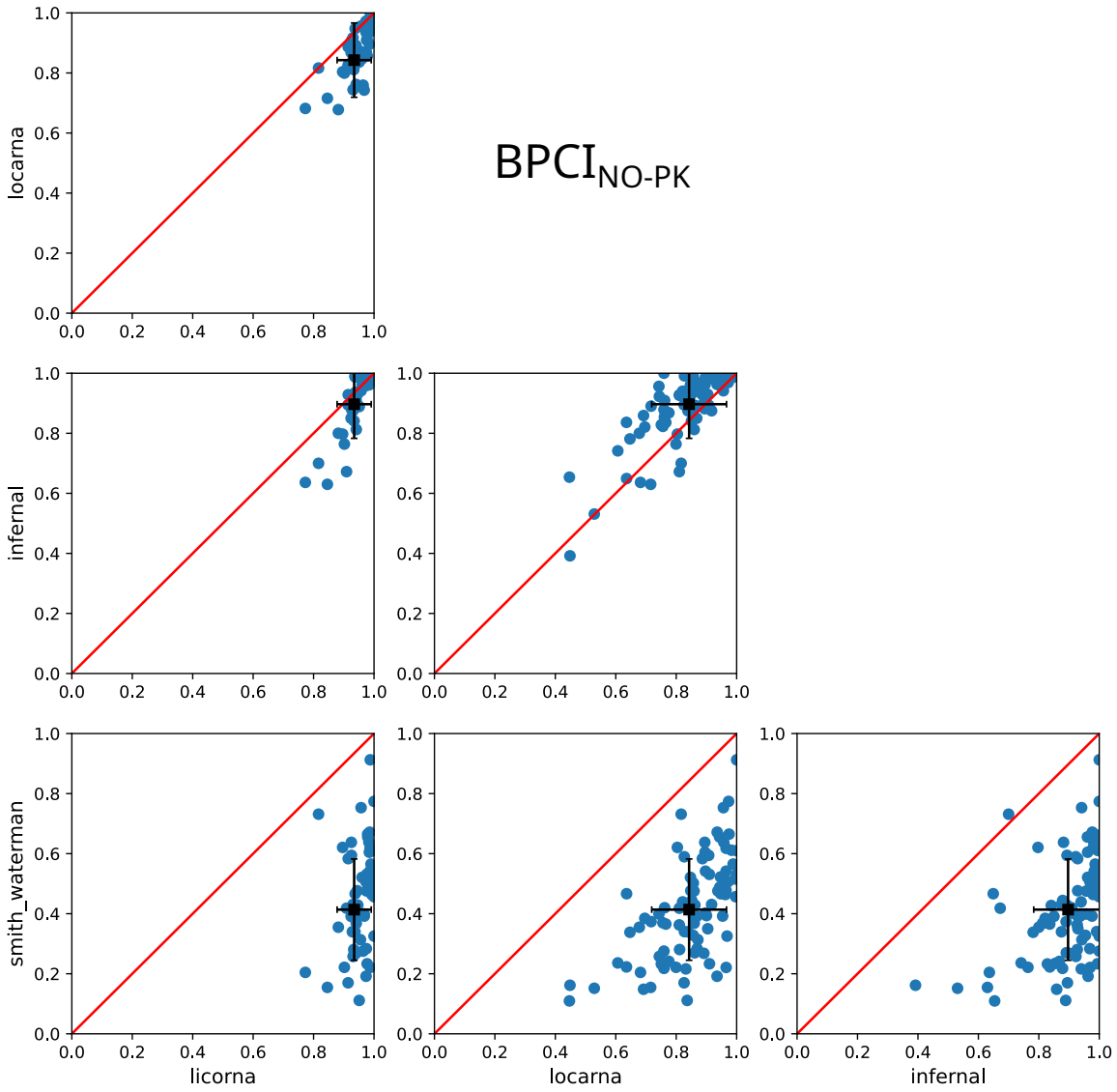


Figure 7: $\text{BPCI}_{\text{NO-PK}}$ results for **Experiment 2**.

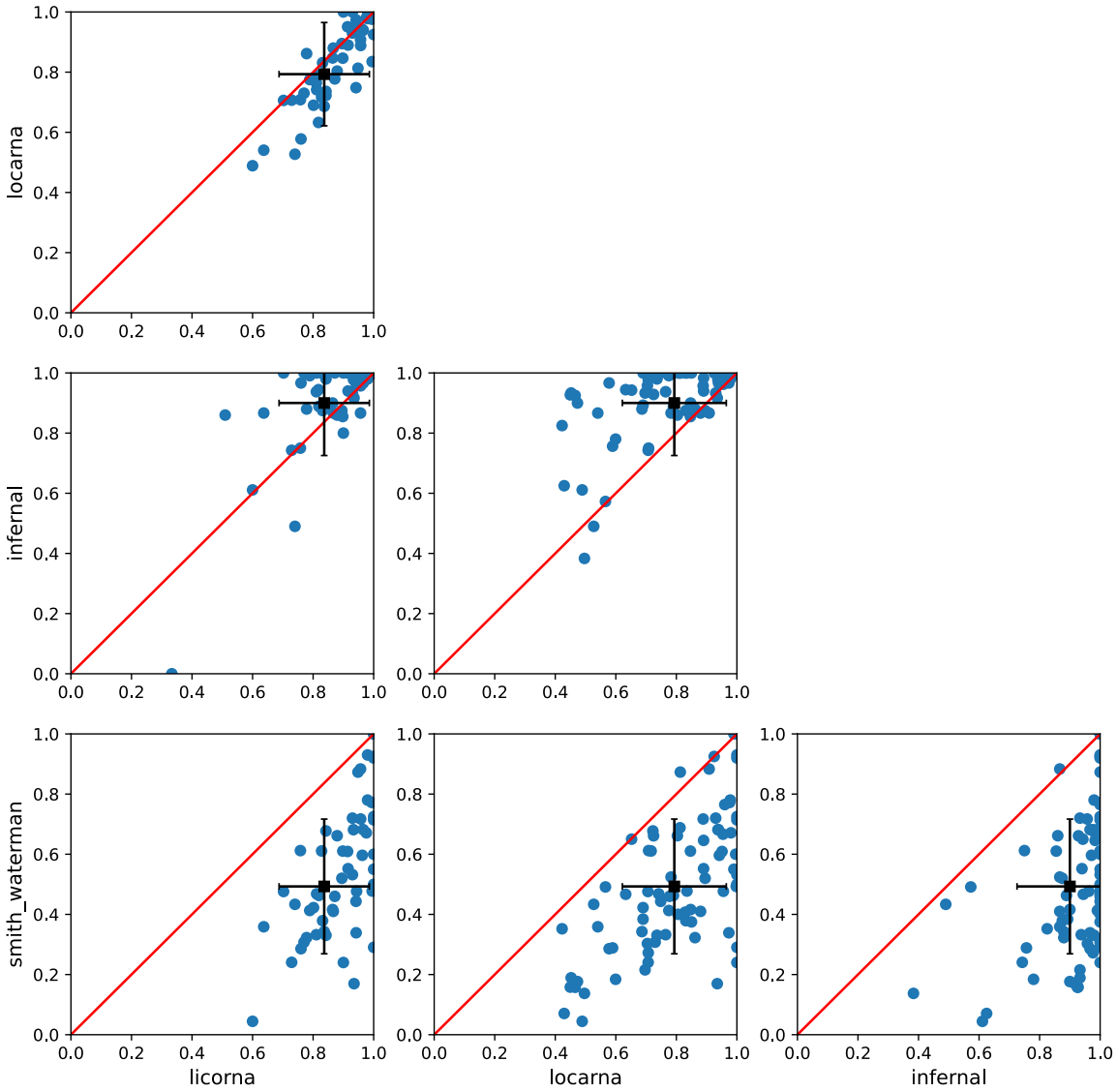


Figure 8: $BPCI_{PK}$ results for **Experiment 2**.

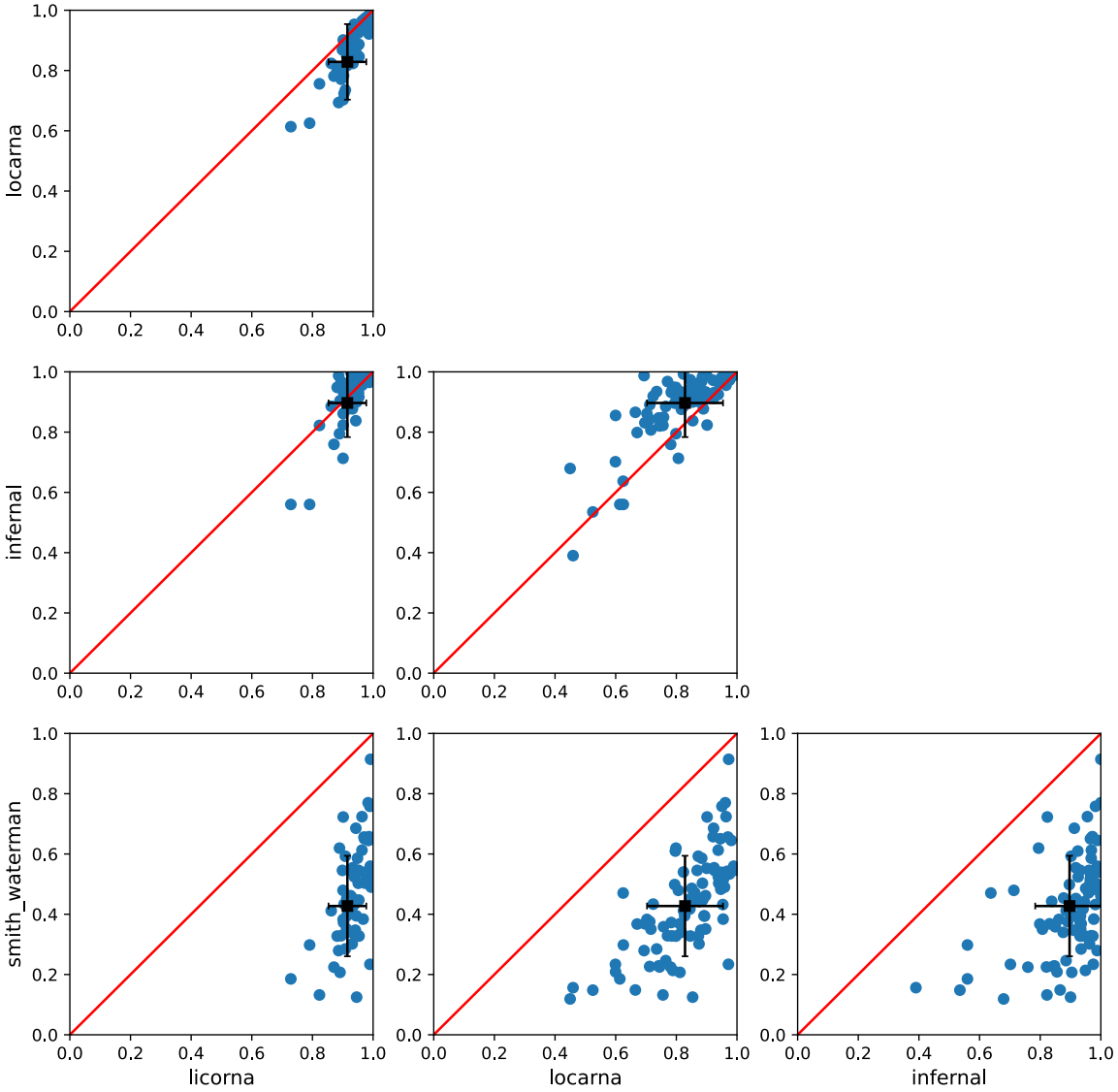


Figure 9: BPCI results for **Experiment 2**.

Proposition 14. *Definition 29 is equivalent to the definition of directed pathwidth from [203] in terms of* vertex separation number*, reading:*

$$dpw(H) = \min_{\sigma \text{ permutation of } V} \max_{\rho \sqsubseteq \sigma} |N^{-1}(\sigma)| \tag{3}$$

for a graph $H = (V, E)$, σ an ordering of its vertices, and ρ a prefix of an ordering σ.

Proof. Given an optimal ordering σ optimizing Equation 3, i.e. the definition of directed pathwidth from [203], and a vertex $u \in V$, define two integers a_u, b_u as follows:

- a_u is the position in σ of the first vertex v such that $u \to v \in E$. If no such vertex exist, then $a_u = b_u$ (see below)
- b_u is the position in σ at which u appears.

Given this definition, for any edge $u \to v$, $a_u \leq b_v$, and $\mathcal{I} = \{[a_u, b_u] \mid u \in V\}$ is therefore a valid interval embedding for H. Let us note $w(\mathcal{I})$ the maximum number of intervals intersecting a given position (i.e. the width of the embedding), and show that $w(\mathcal{I}) - 1 = dpw(H)$.

Let ρ_m be a prefix of σ such that $|N^{-1}(\rho_m)| = dpw(H)$. For any $u \in N^{-1}(\rho_m)|$, by the definition of a_u, b_u, we must have $a_u \leq |\rho_m|$ and $b_u \geq |\rho_m|$. To these vertices whose intervals intersect position $|\rho_m|$ must be added the last element of ρ_m, resulting in $w(\mathcal{I}) - 1 \geq dpw(H)$

Conversely, let i be the position intersecting the most intervals of $\mathcal{I}$. Without loss of generality, we choose it such that it is the end of an interval b_u for some u. By definition of a_u, b_u, the prefix ρ of length b_u is such that $|N^{-1}(\rho)| \geq w(\mathcal{I}) - 1$ and therefore $dpw(H) \geq w(\mathcal{I}) - 1$. Overall $dpw(H) = w(\mathcal{I}) - 1$.

In the other direction, given an optimal interval embedding $\{[a_u, b_u]\}$ of H. An ordering σ can be obtained by taking the vertices in order of increasing b_u value. By similar arguments as above, the width of the interval embedding minus 1 is the directed pathwidth of H. □

D.2 Mixed MIS in bipartite graphs

Our Divide-and-Conquer strategy to the BISR problem relies on the computation of maximum independent sets containing at least one vertex in each part of the input bipartite graph.

We informally call *mixed bipartite maximum independent set* (Mixed-MIS) the problem of deciding whether an input bipartite graph G has a maximum independent set intersecting both of its parts. It is trivially polynomial, as one may check for each pair $(l, r) \in L \times R$, whether $I' \cup \{l, r\}$ is a maximum independent set of G; with I' maximum independent set of G', and G' obtained from G by removing l, r as well as their neighborhoods.

As a maximum independent set of a bipartite graph may be derived from a maximum matching, this simple strategy yield a $O(|V|^2 \cdot \sqrt{|V|}|E|)$ algorithm for our Mixed-MIS problem.

We present here a more efficient strategy, based on a decomposition taking place in two rounds. It results into Algorithm 8. The first round is based on the Dulmage-Mendelsohn decomposition of

bipartite graphs. It yields a partition of the vertices of G into three sets D, A, C, defined as such: for each vertex v of D, there exists a maximum matching in which v is not matched, $A = N(D)$ is the union of the neighborhoods of the vertices of D, and $C = V \setminus (D \cup A)$ contains the remaining vertices. D, A, C verify the following result:

Theorem 22 (Dulmage-Mendelsohn decomposition, Proposition 2.1 of [214], theorem 3.2.4 of [213]). *Given G bipartite graph and D, A, C defined as above, we have that:*

a.
- *D is the intersection of all maximum independent sets of G.*
- *A is the intersection of all minimum vertex covers of G.*
- *the subgraph $G[C]$ induced by C has a perfect matching, which may be deduced from restricting any maximum matching of G to C.*

b. In addition, D may be computed from any maximum matching M of G using the following characterization ([214], lemma 2.2): $D = \overline{W}$ where $\overline{W}$ is composed of the vertices left unmatched by M, as well as all vertices connected to an unmatched vertex through an alternating path of even length.

This decomposition may allow to conclude in some cases (see Algorithm 8). In general, however, a second round of decomposition is needed. In this second round, the set C, which allows for a perfect matching M, is further decomposed into *elementary sub-graphs* (section 4.1 of [213], theorem 4.1.1 and exercise 4.1.5) and [223]. It consists in computing the strongly connected components of a directed graph $H(M, C)$ associated to M and C (same construction as in Section 5.3). The vertices of H are the edges of the matching, and $(l, r) \to (l', r')$ iff l is connected to r' in C. The strongly connected components of H constitute a decomposition of G into elementary sub-graphs. A bipartite graph is elementary iff the sides L, R are the only minimum vertex covers/maximum independent sets [213](theorem 4.1.1). If it is not elementary, then a mixed maximum independent set may be obtained by ordering the elementary sub-graphs $\{(L_i, R_i)\}_{1 \leq i \leq p}$ along a topological order induced by $H(C, M)$. Any set of the form $(\cup_{i \leq t} R_i) \cup (\cup_{i > t} L_i)$ for some $t > 1$ is then a mixed maximum independent set of C.

The discussion above results in Algorithm 8, whose run-time is dominated by the computation of maximum matching in $O(\sqrt{|V|}|E|)$.

D.3 Delayed proofs

D.4 Making an interval representation nice

Let $\{(a_u, b_u) \mid u \in V\}$ be an interval representation for a directed graph H with vertex set V. We explain here how to turn it into a nice interval representation:

If an integer n is such that $a_{u_0} = \cdots = a_{u_l} = b_{v_0} = \cdots = b_{v_p} = n$, we may modify the representation as such:

Algorithm 8 Mixed bipartite maximum independent set

Input: a bipartite graph G with sides L and R. We suppose w.l.o.g that $|L| \geq |R|$.
Output: If it exists, a Maximum Independent Set I of G intersecting both L and R.

M = MaximumMatching(G) ▷ $O(\sqrt{|V|} \cdot |E|)$

I = MaximumIndependentSet(G, M) ▷ $O(|E|)$
if ($I \cap L \neq \emptyset$) ***and*** ($I \cap R \neq \emptyset$) **then**
    return I
end if ▷ // Now $|I| = \max(|L|, |R|)$ and $I = L$ or $I = R$

$D, A, C =$ coarseDulmageMendelsohn(M, G) ▷ $O(|E|)$
if $|L| > |R|$ **then**
    if $R \setminus A \neq \emptyset$ **then**
        ▷//A is the intersection of all minimum vertex covers
        pick $r \in R \setminus A$
        $G' = G \setminus \{r \cup N(r)\}$
        $M' =$ MaximumMatching(G')
        $I' =$ MaximumIndependentSet(G', M')
        return $I' \cup \{r\}$
    else
        return $\perp$; ▷ // Not possible, L is the only MIS
    end if
else if $|L| = |R|$ **then**
    ▷ // L and R are two MIS. So necessarily $D = \emptyset$, $A = \emptyset$, $C = G$
    $\{(L_i, R_i)\}_{1 \leq i \leq p} =$ fineDulmageMendelsohn(M, C) ▷ $O(|V|^2)$
    if p=1 **then**
        return $\perp$
    else
        ▷ Topological sort of the SCCs of H
        s=TopologicalSort($\{(L_i, R_i)\}$) ▷ $O(|V| + |E|)$
        $(L_i, R_i) = s[0]$ ▷ // first in topological sort
        return $R_i \cup (\cup_{j \neq i} L_j)$
    end if
end if

- Interval bounds associated to integers $> n$ are increased by $p+l-1$, to make room for "spreading" $a_{u_1} \dots a_{u_\ell}, b_{v_1} \dots b_{v_p}$.
- $\forall i$, a_{u_i} is set to $n+i$ and b_{v_i} to $l+i$.

None of these modifications change the way intervals intersect one another, leaving the width unchanged. The representation is then "packed" into $[1 \dots 2.|V(H)|]$ by taking the interval bounds in order and setting them to their final position.

D.5 Proof of Proposition 9:

D.6 Re-derivation of Tamaki's algorithm for directed pathwidth

For completeness, we include here a re-derivation of the results of [189], with the slight modification mentioned in the main text related to *pruning*. It results in an algorithm with a $O(n^{\rho+2})$ complexity, slightly different from the $O(n^{\rho+1})$ announced in [189]. The re-derivation follows the same strategy as in the original article, and re-uses most of the notations.

D.7 Commitment lemma - shortest non-expanding extensions (SNEKFEs)

Notations and definitions. In a directed graph, $d^-(u)$ denotes the in-degree of a node u. We work with layouts of vertices, i.e. ordered sequences of vertices, not necessarily containing all vertices. A partial layout σ is called *feasible/valid* if $\forall$ prefix p of σ we have $d^-(p) = |N^-(p)| \leq k$. A partial layout which is *completable* into a valid full layout (for the entire digraph G) is called *strongly feasible* or just *completable into a full solution.* An *extension* τ of σ is a valid partial layout with σ as one of its prefixes. A *shortest non-expanding extension* of σ is an extension τ such that $d^-(\tau) \leq d^-(\sigma)$ and $\forall \rho$ s.t.$V(\sigma) \subsetneq V(\rho) \subsetneq V(\tau)$, $d^-(\rho) > d^-(\sigma)$. In the rest of this note, we will write SNEKFE for shortest non-expanding extension.

Lemma 1 - Commitment Lemma - shortest non-expanding extensions. *If σ is completable into a full solution, and τ is a SNEKFE of σ, then τ is also completable into a full solution.*

In fact, a more general version is true: ρ could be allowed to be equal in d^- to τ before rising again. The proof relies on the fact that, for any two subsets X, Y of vertices of G:

$$d^-(X \cup Y) + d^-(X \cap Y) \leq d^-(X) + d^-(Y)$$

Proof. If σ is completable into a full solution, then $\exists F$ such that $\sigma \cdot F$ is a valid layout for G. Let us reshuffle F into $(\tau \setminus \sigma) \cdot F'$. Within both parts, the ordering of elements is the same as in F. $\tau \cdot F'$ is now a complete layout for G. Is it valid ?

Consider a prefix P of $\tau \cdot F'$. If P is contained within τ, $d^-(P) \leq k$ by the validity of τ.

Else, if P contains some of F', then $P = P' \cup \tau$ for P' a certain prefix of $\sigma \cdot F$. As for $P' \cap \tau$, which we call ρ it verifies $V(\sigma) \subset V(\rho) \subset V(\tau)$ and therefore $d^-(\rho) \geq d^-(\sigma) \geq d^-(\tau)$ by definition of a SNEKFE, with the equality only potentially happening if $\rho = \sigma$ or $\rho = \tau$.

We therefore have:

$$\begin{aligned} d^-(P) &= d^-(P' \cup \tau) \\ &\leq d^-(P') + d^-(\tau) - d^-(\rho) \\ &\leq d^-(P') \leq k \end{aligned}$$

$\tau \cdot F'$ is therefore a valid complete layout for G, and τ is completable into a full solution. □

Let us now describe more precisely what SNEKFEs might look like. We show that they can only be of three types, and formalize it into the next lemma. Its proof relies on the fact that, by adding a single vertex u to a partial layout σ, we may only decrease $d^-(\sigma)$ by at most 1, since $d^-(\sigma) = |N^-(\sigma)|$. We obtain this decrement of 1 if u is a predecessor to a vertex of σ, and does not introduce any new predecessor itself when added.

Lemma 2 - SNEKFE types. *a SNEKFE τ of a partial layout σ may only be of three types:*

- *type-(i): single-vertex "decreasing" extension: $\tau = \sigma \cdot u$ for some vertex u and $d^-(\sigma \cdot u) = d^-(\sigma) - 1$*
- *type-(ii): single-vertex "non-decreasing" extension: $\tau = \sigma \cdot u$ for some vertex u and $d^-(\sigma \cdot u) = d^-(\sigma)$*
- *type-(iii): several vertices "shortcut" extension: τ adds strictly more than one vertex to σ and $d^-(\tau) = d^-(\sigma)$.*

Proof. For single vertex extensions, the two possible types follow from the observation above that the addition of one vertex to a layout can only decrease d^- by at most 1.

For SNEKFEs composed of more than one vertex, observe that if $d^-(\tau) < d^-(\sigma)$, then by considering the prefix ρ of τ obtained by removing just 1 vertex to τ, we would have $d^-(\rho) \leq d^-(\tau) + 1 \leq d^-(\sigma)$. This stems from the observation above that d^- may only decrease by at most 1 when adding a vertex. ρ would be a non-expanding extension of σ shorter than τ, yielding a contradiction. □

D.8 Algorithm

In this section, we restrict ourselves to a pure description of the algorithm, delaying the justification of its correctness and complexity to the "Analysis" section below.

Tree of prefixes (trie). We will build a tree of prefixes of all possible layouts. We prune the tree during its construction thanks to the commitment lemma, as justified in the next section. We call S_i the i^{th} level of the tree of prefixes. I.e. the elements of the tree of length i. $S_0 = \{\emptyset\}$.

Algorithm. S_{i+1} is generated in the following way given S_i:

For each $\sigma \in S_i$:

1. We generate all *feasible immediate extensions* to σ and add them to the tree. I.e the node σ now has the following children set: $\{\sigma \cdot u \quad s.t \quad d^-(\sigma \cdot u) \leq k\}$

2. If some of these immediate extensions verify $d^-(\sigma \cdot u) \leq d^-(\sigma)$, then they are SNEKFEs of σ. In that case, we do the following:

   a. We choose 1 arbitrarily and prune the others.

   b. If the chosen element verifies $d^-(\sigma \cdot u) = d^-(\sigma) - 1$ (the only possibility if $d^-(\sigma \cdot u) < d^-(\sigma)$), then we in addition look for a prefix η of σ verifying $d^-(\eta) = d^-(\sigma \cdot u)$ and $d^-(\rho) > d^-(\eta)\ \forall \rho$ s.t. $\eta \sqsubseteq \rho \sqsubseteq \sigma \cdot u$, $\rho \neq \eta$, $\rho \neq \sigma \cdot u$.
   If such an η is found, then any part of tree branching off the path from η to $\sigma \cdot u$ is removed. Note that this might shorten the overall loop over $\sigma \in S_i$.

End Algorithm

D.9 Analysis

This section will be composed of three parts. In the first one, we define an invariant property ("internally pruned") for trees of prefixes of layouts of vertices. In the second one, we show that, in the algorithm presented in the previous section, the tree of prefixes verifies the invariant at all times, and prove the correctness of the algorithm. Finally, in the third part, we analyze the size of trees of prefixes verifying the invariant, proving that each level S_i of such a tree has a size $\leq n^k$, yielding a complexity analysis of the algorithm.

D.9.1 Internally pruned trees of prefixes

Definition - Internally pruned. *A tree $\mathcal{T}$ of prefixes of layouts of vertices (such as the one used in the algorithm in the previous section) is said to be internally pruned if for all pairs (σ, τ) of nodes of $\mathcal{T}$ such that τ is a shortest non-expanding extension of σ, all nodes on the path from τ (included) to σ (excluded) in $\mathcal{T}$ have degree exactly 2. I.e. there are no sub-parts of the tree rooted on the path from τ (included) to σ (excluded)*

We use the term "internally" to emphasize the fact that, in a context where we apply the definition of "internally pruned" to a partially constructed $\mathcal{T}$ within the algorithm of the previous section, More ("external") pruning of the tree might be achieved further in the construction of the tree, as new SNEKFEs are discovered (see below for the justification of why new SNEKFEs are indeed discovered at step 2.b of the algorithm).

D.9.2 Invariant and correctness

Lemma 3 - Invariant. *Throughout the execution of the algorithm presented in the previous section, the tree $\mathcal{T}$ of prefixes of layouts of vertices remains "internally pruned" at all times*

Proof. The tree $\mathcal{T}$ starts off with one node for the empty sequence. It is therefore internally pruned.

Suppose now that the tree of prefixes $\mathcal{T}$ is internally pruned at an intermediate step in the algorithm, then the next building step always consists in considering a leaf σ and executing step 1. and 2. of the algorithm. Several cases may arise:

- If all of the immediate extensions are such that $\{d^-(\sigma) < d^-(\sigma \cdot u) \leq k\}$, then no new SNEKFEs are generated when adding them to the tree. (if $\sigma \cdot u$ is a SNEKFE of some η up the tree, then σ is shorter and also non-expanding). After the addition of the immediate extension, the tree is therefore still internally pruned.

- If one of these immediate extensions verifies $d^-(\sigma \cdot u) = d^-(\sigma)$ but none of them verify $d^-(\sigma \cdot u) < d^-(\sigma)$, then one of these extensions is a SNEKFE of σ, and is kept while the others are pruned. However, this is the only SNEKFE introduced by the extension. Therefore, the pruning of immediate extensions other than the selected one is enough to keep the tree internally pruned.

- If one of the immediate extensions verifies $d^-(\sigma \cdot u) = d^-(\sigma) - 1$, then one of the immediate extensions is selected and the others are pruned, as in the previous case. However, in addition, $\sigma \cdot u$ might be a new shortest non-expanding extension of a node η up the tree.

  If this is the case, then there is only one such η, per the definition of shortest non-expanding extensions.

  We argue that the conditions used in the algorithm indeed detect such an η.

  If $\sigma \cdot u$ is a SNEKFE of η, then the conditions described in the algorithm (that $d^-(\sigma \cdot u) = d^-(\eta)$, and $d^-(\rho) > d^-(\eta)$ for any ρ on the path from η to $\sigma \cdot u$) are verified.

  Conversely, if the conditions are verified, then suppose η has a shorter non-expanding extensions τ. τ cannot be on the path from η to $\sigma \cdot u$ as that would imply $d^-(\tau) > d^-(\eta)$. Since τ is shorter than $\sigma \cdot u$, τ has been generated in a previous step of the algorithm. At this point, step $2.b$ of the algorithm would have pruned the path to σ, which cannot be visited, leading to a contradiction.

  Therefore, the potentially newly introduced SNEKFE is detected, and the corresponding pruning is carried out, leaving the tree internally pruned

Therefore, after each extension of the tree throughout the algorithm, the tree remains internally pruned. □

We quickly finish this sub-section with a proof of correctness of the algorithm.

Lemma 4 - correctness. *If the graph G allows for a full k-feasible solution, then there is such a solution among the leaves of the tree of prefixes $\mathcal{T}$ generated by the algorithm.*

Proof. Denote the set of full solutions S, and suppose all solutions are absent from $\mathcal{T}$.

$\forall \sigma \in S$, there is some (possibly empty) prefix of σ in $\mathcal{T}$.

We pick $\sigma \in S$ allowing for the largest prefix $\eta \in \mathcal{T}$, i.e:

$$\sigma = \underset{\sigma' \in S}{\text{argmax}} \left[\max_{\eta \sqsubseteq \sigma', \eta \in \mathcal{T}} |\eta| \right]$$

Take η the largest prefix of σ belonging to $\mathcal{T}$. If the path from η to σ has been pruned, it is because η is on the path from η' to τ, with τ shortest non expanding extension of η', and τ is not a prefix of σ.

The path from η to σ is pruned only when τ is visited. Hence $\tau \in \mathcal{T}$, otherwise, the path from

Per the commitment lemma, τ is the prefix of a full solution σ''. But $|\tau| > |\eta|$, contradicting the choice of σ. □

D.9.3 Signature analysis

We show here that, at any point in the algorithm, thanks to the pruning, $\forall i, \quad |S_i| = O(n^k)$.

Definition - signature . Consider $\sigma \in S_i$ for some i, within the internally pruned tree generated by the algorithm, valid partial layout. We call *signature* of σ the set of vertices obtained from $V(\sigma)$ by removing, given any pair (η, ρ) of prefixes of σ such that ρ is a SNEKFE of η, all vertices in $\rho \setminus \eta$.

Given $\sigma \in S_i$, its signature can be easily computed by looking at the path from the root to σ: any vertex chosen out of several available possibilities is part of the signature, while any vertex that was the only possibility at the point of its choosing isn't.

Lemma 5 - Same signature same sequence. *If $sgn(\sigma) = sgn(\tau)$ within the pruned tree of layouts and $|\tau| = |\sigma|$ then $\sigma = \tau$*

Proof. When starting at the root and building τ and σ by going down the tree, at every node, there are two cases:

- Either the next move is part of a SNEKFE. In this case there are no choices to be made, the added vertex is not part of the signature, and is the same for σ and τ.
- Or the next move is not part of a SNEKFE. In this case, several choices are possible, and the next added vertex will be part of the signature. Since the signatures of σ and τ are the same, the same vertex is added to σ and τ.

At the end of this process, σ and τ are therefore identical. □

Lemma 6 - overall strictly decreasing = SNEKFE only. *Consider $\tau \in S_i$ for some i partial valid layout, and σ a prefix of τ such that:*

- $d^-(\sigma) > d^-(\tau)$
- *For any ρ such that $\sigma \sqsubseteq \rho \sqsubseteq \tau$, $\rho \neq \tau$, we also have $d^-(\rho) > d^-(\tau)$.*

Then, the suffix $\tau \setminus \sigma$ of τ corresponding to σ can be entirely partitioned into SNEKFEs. In particular, none of its elements are part of the signature of τ.

Proof. We prove the lemma by induction on the length of the suffix $\tau \setminus \sigma$. If $|\tau \setminus \sigma| = 1$, then $\tau = \sigma \cdot u$ and $d^-(\tau) = d^-(\sigma) - 1$. τ is a type-(i) SNEKFE of σ and the lemma is true.

If $|\tau \setminus \sigma| > 1$ and we assume the lemma true $\forall l < |\tau \setminus \sigma|$, then let us distinguish two cases related to the first element v of $\tau \setminus \sigma$:

- if $\sigma \cdot v$ is a type-(i) or type-(ii) SNEKFE of σ, then we apply the induction hypothesis to the suffix $\tau \setminus (\sigma \cdot v)$ of τ and we have the result.

- else, if $d^-(\sigma \cdot v) > d^-(\sigma)$, we know, since $d^-(\tau) < d^-(\sigma)$ and the d^--curve only decreases by steps of -1, that there must exist ρ such that $d^-(\rho) = d^-(\sigma)$, $\sigma \sqsubseteq \rho \sqsubseteq \tau$, and $d^-(\rho') > d^-(\sigma)$ for any ρ' such that $\sigma \sqsubseteq \rho' \subseteq \rho$ (ρ is the shortest prefix of τ which contains σ and has the same d^- value). ρ is then a type-(iii) SNEKFE of σ by Lemma 4, and we may apply the induction hypothesis to $\tau \setminus \sigma$

□

Lemma 7 - Signature size. *$\forall \sigma \in S_i$ for some i partial layout of vertices, $|sgn(\sigma)| \leq d^-(\sigma)$*

Proof. The proof is by induction on $|\sigma|$. Suppose $|sgn(\sigma)| \leq d^-(\sigma)$, and consider the extension $\sigma \cdot u$, where u is a vertex.

- If $\sigma \cdot u$ is not a SNEKFE of σ, then $|sgn(\sigma \cdot u)| = |sgn(\sigma) \cup \{u\}| = |sgn(u)| + 1 \leq d^-(\sigma) + 1 \leq d^-(\sigma \cdot u)$
- If σ is a type-(ii) SNEKFE of σ, then $sgn(\sigma) = sgn(\sigma \cdot u)$ and $d^-(\sigma \cdot u) = d^-(\sigma)$.
- If $\sigma \cdot u$ is a type-(i) SNEKFE of σ, then consider η, the closest node (up the tree) such that $d^-(\eta) < d^-(\sigma \cdot u)$, and $\eta \cdot v$ its successor on the path to $\sigma \cdot u$. We have $d^-(\eta) < d^-(\sigma \cdot u) \leq d^-(\eta \cdot v)$, by definition of η. The path from $\eta \cdot v$ to u is either a type-(iii) SNEKFE or overall-decreasing. Therefore $sgn(\sigma \cdot u) = sgn(\eta \cdot v)$. and $|sgn(\sigma \cdot u)| = |sgn(\eta)| + 1 \leq d^-(\eta) + 1$ by induction hypothesis, and $|sgn(\sigma \cdot u)| \leq d^-(\sigma \cdot u)$.

□

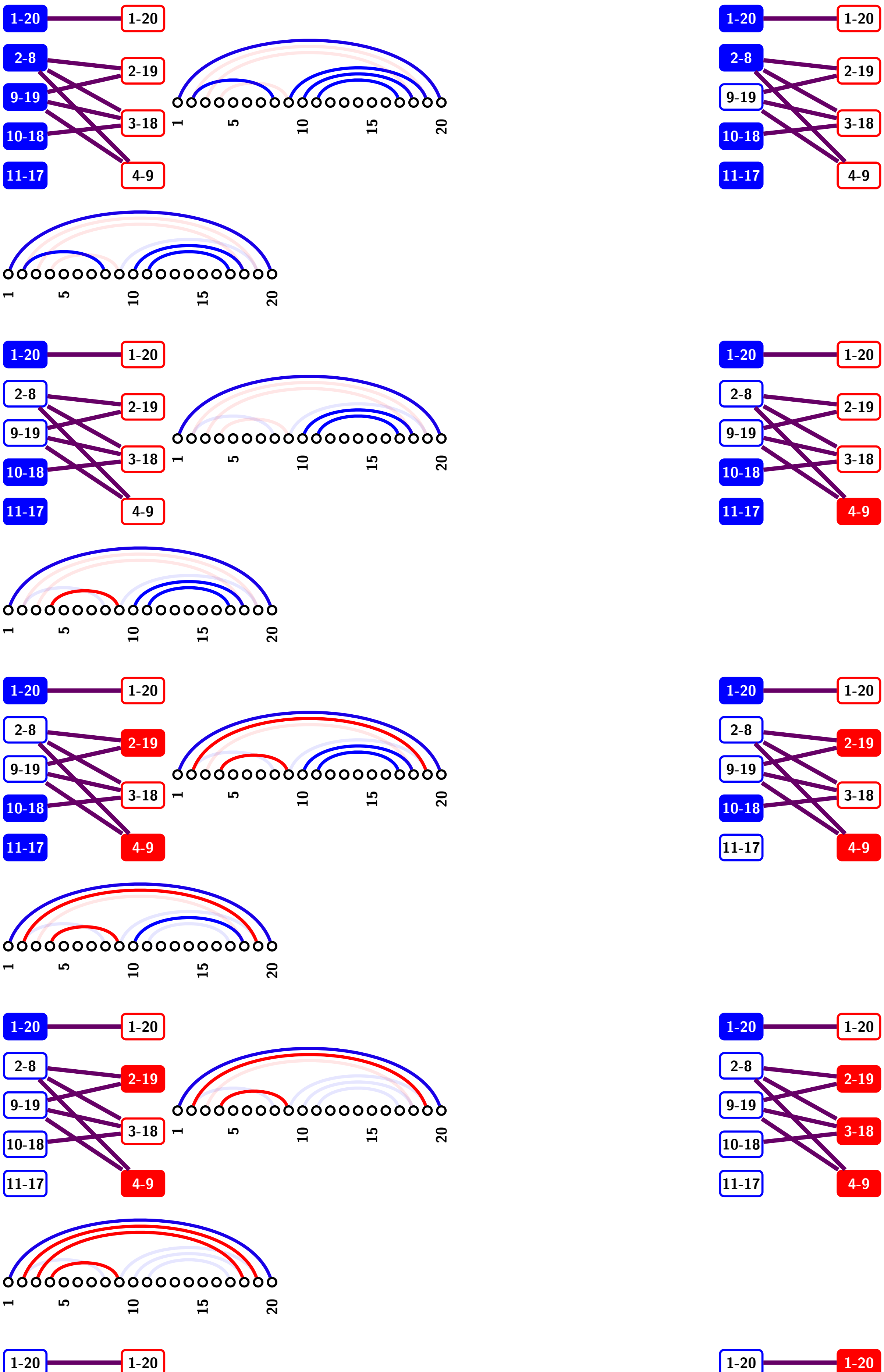
In particular, $\forall \sigma$ partial layout, $d^-(\sigma) \leq k$. Since two different elements of S_i need different signatures, we get the following corollary:

Corollary. *$\forall i$, at any point in the algorithm, $|S_i| = O(n^k)$*

The overall complexity of the algorithm is therefore $O(n^{k+O(1)})$. More precisely, it is $O(n^{k+2})$. (there are n levels of the tree to fill, $\leq n^k$ nodes per level and $O(n)$ work per node to generate the next level).

D.10 Detailed RNA reconfiguration example

We provide in Figure 10 the intermediate sets of base pairs, and associated RNA secondary structures, for our running example, described in Figures 5.1.


1-20
2-8
9-19
10-18
11-17
1-20
2-19
3-18
4-9
1
5
10
15
20

Bibliography

[1] Thomas R Cech. The RNA worlds in context. *Cold Spring Harbor perspectives in biology*, 4(7):a006742, 2012.

[2] Yiran Zhu, Liyuan Zhu, Xian Wang, and Hongchuan Jin. RNA-based therapeutics: An overview and prospectus. *Cell Death & Disease*, 13(7):644, 2022.

[3] Tulsi Ram Damase, Roman Sukhovershin, Christian Boada, Francesca Taraballi, Roderic I Pettigrew, and John P Cooke. The limitless future of RNA therapeutics. *Frontiers in bioengineering and biotechnology*, page 161, 2021.

[4] Rein Verbeke, Ine Lentacker, Stefaan C De Smedt, and Heleen Dewitte. Three decades of messenger RNA vaccine development. *Nano Today*, 28:100766, 2019.

[5] Neocles B Leontis and Eric Westhof. Geometric nomenclature and classification of RNA base pairs. *Rna*, 7(4):499–512, 2001.

[6] Ignacio Tinoco Jr and Carlos Bustamante. How RNA folds. *Journal of molecular biology*, 293(2):271–281, 1999.

[7] Alexander F Palazzo and Eliza S Lee. Non-coding RNA: what is functional and what is junk? *Frontiers in genetics*, 6:2, 2015.

[8] John S Mattick and Igor V Makunin. Non-coding RNA. *Human molecular genetics*, 15(suppl_1):R17–R29, 2006.

[9] Divyaa Bhagdikar, Frank J Grundy, and Tina M Henkin. Transcriptional and translational S-box riboswitches differ in ligand-binding properties. *Journal of Biological Chemistry*, 295(20):6849–6860, 2020.

[10] Yijin Liu, Timothy J Wilson, Scott A McPhee, and David MJ Lilley. Crystal structure and mechanistic investigation of the twister ribozyme. *Nature chemical biology*, 10(9):739–744, 2014.

[11] Ioanna Kalvari, Eric P Nawrocki, Nancy Ontiveros-Palacios, Joanna Argasinska, Kevin Lamkiewicz, Manja Marz, Sam Griffiths-Jones, Claire Toffano-Nioche, Daniel Gautheret, Zasha

Weinberg, et al. Rfam 14: expanded coverage of metagenomic, viral and microRNA families. *Nucleic Acids Research*, 49(D1):D192–D200, 2021.

[12] Andrew M Waterhouse, James B Procter, David MA Martin, Michèle Clamp, and Geoffrey J Barton. Jalview Version 2—a multiple sequence alignment editor and analysis workbench. *Bioinformatics*, 25(9):1189–1191, 2009.

[13] Yann Ponty and Vladimir Reinharz. RNA Folding. *From Sequences to Graphs: Discrete Methods and Structures for Bioinformatics*, pages 185–231, 2022.

[14] Édouard Bonnet, Paweł Rzążewski, and Florian Sikora. Designing RNA secondary structures is hard. *Journal of Computational Biology*, 27(3):302–316, 2020.

[15] Chris Thachuk, Jan Manuch, Arash Rafiey, Leigh-Anne Mathieson, Ladislav Stacho, and Anne Condon. An Algorithm for the Energy Barrier Problem Without Pseudoknots and Temporary Arcs. In *Biocomputing 2010*, pages 108–119. World Scientific, oct 2009.

[16] 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.

[17] A Xayaphoummine, T Bucher, F Thalmann, and H Isambert. Prediction and statistics of pseudoknots in RNA structures using exactly clustered stochastic simulations. *Proceedings of the National Academy of Sciences*, 100(26):15310–15315, 2003.

[18] 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.

[19] Michael Zuker, David H Mathews, and Douglas H Turner. Algorithms and thermodynamics for RNA secondary structure prediction: a practical guide. *RNA biochemistry and biotechnology*, pages 11–43, 1999.

[20] John S McCaskill. The equilibrium partition function and base pair binding probabilities for RNA secondary structure. *Biopolymers: Original Research on Biomolecules*, 29(6-7):1105–1119, 1990.

[21] Wilma K Olson, Shuxiang Li, Thomas Kaukonen, Andrew V Colasanti, Yurong Xin, and Xiang-Jun Lu. Effects of noncanonical base pairing on RNA folding: structural context and spatial arrangements of G· A pairs. *Biochemistry*, 58(20):2474–2487, 2019.

[22] Xiang-Jun Lu, Harmen J Bussemaker, and Wilma K Olson. DSSR: an integrated software tool for dissecting the spatial structure of RNA. *Nucleic acids research*, 43(21):e142–e142, 2015.

[23] Vladimir Reinharz, Roman Sarrazin-Gendron, and Jérôme Waldispühl. Modeling and predicting rna three-dimensional structures. *RNA Bioinformatics*, pages 17–42, 2021.

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

[25] 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.

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

[27] 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.

[28] Douglas H Turner and David H Mathews. NNDB: the nearest neighbor parameter database for predicting stability of nucleic acid secondary structure. *Nucleic acids research*, 38(suppl_1):D280–D282, 2010.

[29] 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.

[30] 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.

[31] Mirela S Andronescu, Cristina Pop, and Anne E Condon. Improved free energy parameters for RNA pseudoknotted secondary structure prediction. *Rna*, 16(1):26–42, 2010.

[32] Jack E. Tabaska, Robert B. Cary, Harold N. Gabow, and Gary D. Stormo. An RNA folding method capable of identifying pseudoknots and base triples. *Bioinformatics (Oxford, England)*, 14(8):691–699, 1998.

[33] Rune B Lyngsø. Complexity of pseudoknot prediction in simple models. In *International Colloquium on Automata, Languages, and Programming*, pages 919–931. Springer, 2004.

[34] Saad Sheikh, Rolf Backofen, and Yann Ponty. Impact of the energy model on the complexity of RNA folding with pseudoknots. In *Combinatorial Pattern Matching: 23rd Annual Symposium, CPM 2012, Helsinki, Finland, July 3-5, 2012. Proceedings 23*, pages 321–333. Springer, 2012.

[35] Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, and Clifford Stein. *Introduction to algorithms*. MIT press, 2022.

[36] Michael S Waterman. Secondary structure of single-stranded nucleic acids. *Adv. math. suppl. studies*, 1:167–212, 1978.

[37] Daniel H Younger. Recognition and parsing of context-free languages in time n3. *Information and control*, 10(2):189–208, 1967.

[38] Itiroo Sakai. Syntax in universal translation. In *Proceedings of the International Conference on Machine Translation and Applied Language Analysis*, 1961.

[39] Jack Edmonds. Maximum matching and a polyhedron with 0, 1-vertices. *Journal of research of the National Bureau of Standards B*, 69(125-130):55–56, 1965.

[40] Grégoire De Bisschop, Delphine Allouche, Elisa Frezza, Benoît Masquida, Yann Ponty, Sebastian Will, and Bruno Sargueil. Progress toward SHAPE constrained computational prediction of tertiary interactions in RNA structure. *Non-coding RNA*, 7(4):71, 2021.

[41] Katherine E Deigan, Tian W Li, David H Mathews, and Kevin M Weeks. Accurate SHAPE-directed RNA structure determination. *Proceedings of the National Academy of Sciences*, 106(1):97–102, 2009.

[42] James K Baker. Trainable grammars for speech recognition. *The Journal of the Acoustical Society of America*, 65(S1):S132–S132, 1979.

[43] Sebastian Will, Tejal Joshi, Ivo L Hofacker, Peter F Stadler, and Rolf Backofen. LocARNA-P: accurate boundary prediction and improved detection of structural RNAs. *Rna*, 18(5):900–914, 2012.

[44] Ronny Lorenz, Stephan H Bernhart, Christian Höner zu Siederdissen, Hakim Tafer, Christoph Flamm, Peter F Stadler, and Ivo L Hofacker. ViennaRNA Package 2.0. *Algorithms for molecular biology*, 6:1–14, 2011.

[45] 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.

[46] Alexander Churkin, Matan Drory Retwitzer, Vladimir Reinharz, Yann Ponty, Jérôme Waldispühl, and Danny Barash. Design of RNAs: comparing programs for inverse RNA folding. *Briefings in bioinformatics*, 19(2):350–358, 2018.

[47] Hua-Ting Yao. *Local decomposition in RNA structural design.* McGill University (Canada), 2021.

[48] Hua-Ting Yao, Jérôme Waldispühl, Yann Ponty, and Sebastian Will. Taming Disruptive Base Pairs to Reconcile Positive and Negative Structural Design of RNA. In *Research in Computational Molecular Biology - 25th international conference on research in computational molecular biology (RECOMB 2021)*, Padova, France, April 2021.

[49] Akito Taneda. Multi-objective optimization for RNA design with multiple target secondary structures. *BMC bioinformatics*, 16:1–20, 2015.

[50] Sherry Y Wu, Gabriel Lopez-Berestein, George A Calin, and Anil K Sood. RNAi therapies: drugging the undruggable. *Science translational medicine*, 6(240):240ps7–240ps7, 2014.

[51] Norbert Pardi, Michael J Hogan, Frederick W Porter, and Drew Weissman. mRNA vaccines—a new era in vaccinology. *Nature reviews Drug discovery*, 17(4):261–279, 2018.

[52] Peter B Dykstra, Matias Kaplan, and Christina D Smolke. Engineering synthetic RNA devices for cell control. *Nature Reviews Genetics*, 23(4):215–228, 2022.

[53] Robert Kleinkauf, Torsten Houwaart, Rolf Backofen, and Martin Mann. antaRNA–Multi-objective inverse folding of pseudoknot RNA using ant-colony optimization. *BMC bioinformatics*, 16(1):1–7, 2015.

[54] Joseph N Zadeh, Conrad D Steenberg, Justin S Bois, Brian R Wolfe, Marshall B Pierce, Asif R Khan, Robert M Dirks, and Niles A Pierce. NUPACK: Analysis and design of nucleic acid systems. *Journal of computational chemistry*, 32(1):170–173, 2011.

[55] Nono SC Merleau and Matteo Smerlak. aRNAque: an evolutionary algorithm for inverse pseudoknotted RNA folding inspired by Lévy flights. *BMC bioinformatics*, 23(1):335, 2022.

[56] Anke Busch and Rolf Backofen. INFO-RNA—a fast approach to inverse RNA folding. *Bioinformatics*, 22(15):1823–1831, 2006.

[57] Marco C Matthies, Stefan Bienert, and Andrew E Torda. Dynamics in sequence space for RNA secondary structure design. *Journal of chemical theory and computation*, 8(10):3663–3670, 2012.

[58] Christian Höner zu Siederdissen, Stefan Hammer, Ingrid Abfalter, Ivo L Hofacker, Christoph Flamm, and Peter F Stadler. Computational design of rnas with complex energy landscapes. *Biopolymers*, 99(12):1124–1136, 2013.

[59] Stefan Hammer, Wei Wang, Sebastian Will, and Yann Ponty. Fixed-parameter tractable sampling for RNA design with multiple target structures. *BMC bioinformatics*, 20:1–13, 2019.

[60] Tao Jiang, Guohui Lin, Bin Ma, and Kaizhong Zhang. A general edit distance between RNA structures. *Journal of computational biology*, 9(2):371–388, 2002.

[61] Eric P Nawrocki and Sean R Eddy. Infernal 1.1: 100-fold faster RNA homology searches. *Bioinformatics*, 29(22):2933–2935, 2013.

[62] 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 Ben Raphael and Jijun Tang, editors, *Algorithms in Bioinformatics*, pages 149–164, Ljubljana, Slovenia, 2012. Springer.

[63] Ján Maňuch, Chris Thachuk, Ladislav Stacho, and Anne Condon. NP-completeness of the energy barrier problem without pseudoknots and temporary arcs. *Natural Computing*, 10(1):391–405, 2011.

[64] Stefan Badelt, Ronny Lorenz, and Ivo L Hofacker. DrTransformer: heuristic cotranscriptional RNA folding using the nearest neighbor energy model. *Bioinformatics*, 39(1):btad034, 2023.

[65] Michael T Wolfinger, W Andreas Svrcek-Seiler, Christoph Flamm, Ivo L Hofacker, and Peter F Stadler. Efficient computation of RNA folding dynamics. *Journal of Physics A: Mathematical and General*, 37(17):4731, 2004.

[66] Marcel Kucharik, Ivo L Hofacker, Peter F Stadler, and Jing Qin. Basin Hopping Graph: a computational framework to characterize RNA folding landscapes. *Bioinformatics*, 30(14):2009–2017, 2014.

[67] Hiroki Takizawa, Junichi Iwakiri, Goro Terai, and Kiyoshi Asai. Finding the direct optimal RNA barrier energy and improving pathways with an arbitrary energy model. *Bioinformatics*, 36(Supplement_1):i227–i235, 2020.

[68] Christoph Flamm, Ivo L Hofacker, Peter F Stadler, and Michael T Wolfinger. Barrier trees of degenerate landscapes. 2002.

[69] Juraj Michálik, Hélène Touzet, and Yann Ponty. Efficient approximations of RNA kinetics landscape using non-redundant sampling. *Bioinformatics*, 33(14):i283–i292, 2017.

[70] Gregor Entzian, Ivo L Hofacker, Yann Ponty, Ronny Lorenz, and Andrea Tanzer. RNAxplorer: harnessing the power of guiding potentials to sample RNA landscapes. *Bioinformatics*, 37(15):2126–2133, 2021.

[71] Marek Cygan, Fedor V Fomin, Łukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michał Pilipczuk, and Saket Saurabh. *Parameterized algorithms*, volume 5. Springer, Cham, 2015.

[72] Lowell W Beineke, Robin J Wilson, and Ortrud R Oellermann. *Topics in structural graph theory*. Cambridge University Press Cambridge, 2013.

[73] Bruno Courcelle. The monadic second-order logic of graphs III: Tree-decompositions, minors and complexity issues. *RAIRO-Theoretical Informatics and Applications-Informatique Théorique et Applications*, 26(3):257–286, 1992.

[74] Joachim Kneis and Alexander Langer. A practical approach to Courcelle's theorem. *Electronic Notes in Theoretical Computer Science*, 251:65–81, 2009.

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

[76] 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.

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

[78] Mahdi Belbasi and Martin Fürer. Finding all leftmost separators of size ≤ k≤ k. In *Combinatorial Optimization and Applications: 15th International Conference, COCOA 2021, Tianjin, China, December 17–19, 2021, Proceedings 15*, pages 273–287. Springer, 2021.

[79] Tuukka Korhonen. A single-exponential time 2-approximation algorithm for treewidth. In *2021 IEEE 62nd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 184–192. IEEE, 2022.

[80] Tuukka Korhonen and Daniel Lokshtanov. An improved parameterized algorithm for treewidth. In *Proceedings of the 55th Annual ACM Symposium on Theory of Computing*, pages 528–541, 2023.

[81] Hans L Bodlaender and Arie MCA Koster. Treewidth computations I. Upper bounds. *Information and Computation*, 208(3):259–275, 2010.

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

[83] Holger Dell, Christian Komusiewicz, Nimrod Talmon, and Mathias Weller. The PACE 2017 parameterized algorithms and computational experiments challenge: The second iteration. In *12th international symposium on parameterized and exact computation (IPEC 2017)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2018.

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

[85] Neil Robertson and Paul D Seymour. Graph minors. XX. Wagner's conjecture. *Journal of Combinatorial Theory, Series B*, 92(2):325–357, 2004.

[86] Neil Robertson and Paul D Seymour. Graph minors. XIII. The disjoint paths problem. *Journal of combinatorial theory, Series B*, 63(1):65–110, 1995.

[87] Yinglei Song, Chunmei Liu, Russell Malmberg, Fangfang Pan, and Liming Cai. Tree decomposition based fast search of RNA structures including pseudoknots in genomes. In *2005 IEEE Computational Systems Bioinformatics Conference (CSB'05)*, pages 223–234. IEEE, 2005.

[88] H. M. Berman, J. Westbrook, Z. Feng, G. Gilliland, T. N. Bhat, H. Weissig, I. N. Shindyalov, and P. E. Bourne. The Protein Data Bank. *Nucleic acids research*, 28:235–242, January 2000.

[89] Thomas van Dijk, Jan-Pieter van den Heuvel, and Wouter Slob. Computing treewidth with LibTW. *Citeseer. http://citeseerx. ist. psu. edu/viewdoc/download*, 2006.

[90] Lusheng Wang and Tao Jiang. On the complexity of multiple sequence alignment. *Journal of computational biology*, 1(4):337–348, 1994.

[91] Daniel H Huson, Regula Rupp, and Celine Scornavacca. *Phylogenetic networks: concepts, algorithms and applications*. Cambridge University Press, 2010.

[92] FR McMorris and Michael A Steel. The complexity of the median procedure for binary trees. In *New Approaches in Classification and Data Analysis*, pages 136–140. Springer, 1994.

[93] Rodney G. Downey and M. R. Fellows. *Parameterized Complexity*. Springer, Berlin, Heiderlberg, 2012.

[94] Hans L. Bodlaender, Rodney G. Downey, Michael R. Fellows, Michael T. Hallett, and Harold T. Wareham. Parameterized complexity analysis in computational biology. *Bioinformatics*, 11(1):49–57, 1995.

[95] Laurent Bulteau and Mathias Weller. Parameterized algorithms in bioinformatics: an overview. *Algorithms*, 12(12):256, 2019.

[96] Chunmei Liu, Yinglei Song, and Louis Shapiro. RNA folding including pseudoknots: A new parameterized algorithm and improved upper bound. In *Algorithms in Bioinformatics: 7th International Workshop, WABI 2007, Philadelphia, PA, USA, September 8-9, 2007. Proceedings 7*, pages 310–322. Springer, 2007.

[97] Yinglei Song, Chunmei Liu, Xiuzhen Huang, Russell L Malmberg, Ying Xu, and Liming Cai. Efficient parameterized algorithms for biopolymer structure-sequence alignment. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 3(4):423–432, 2006.

[98] Bertrand Marchand, Yann Ponty, and Laurent Bulteau. Tree diet: reducing the treewidth to unlock FPT algorithms in RNA bioinformatics. *Algorithms for Molecular Biology*, 17(1):1–17, 2022.

[99] Bertrand Marchand, Yann Ponty, and Laurent Bulteau. Tree Diet: Reducing the Treewidth to Unlock FPT Algorithms in RNA Bioinformatics. In *21st International Workshop on Algorithms in Bioinformatics*, volume 7, page 118, 2021.

[100] Bertrand Marchand, Sebastian Will, Sarah Berkemer, Laurent Bulteau, and Yann Ponty. Automated design of dynamic programming schemes for RNA folding with pseudoknots. *Algorithms for Molecular Biology*, 2023.

[101] Laurent Bulteau, Bertrand Marchand, and Yann Ponty. A new parametrization for independent set reconfiguration and applications to RNA kinetics. In *IPEC 2021-International Symposium on Parameterized and Exact Computation*, 2021.

[102] Théo Boury, Laurent Bulteau, Bertrand Marchand, and Yann Ponty. Independent set reconfiguration: general and RNA-focused parameterized algorithms. 2023.

[103] Bertrand Marchand, Sebastian Will, Sarah J Berkemer, Laurent Bulteau, and Yann Ponty. Automated Design of Dynamic Programming Schemes for RNA Folding with Pseudoknots. In *22nd International Workshop on Algorithms in Bioinformatics*, 2022.

[104] Mathias Weller, Annie Chateau, and Rodolphe Giroudeau. Exact approaches for scaffolding. *BMC Bioinformatics*, 16(S14), oct 2015.

[105] Jinbo Xu. Rapid protein side-chain packing via tree decomposition. In *Research in Computational Molecular Biology (RECOMB 2005)*, volume 3500 of *Lecture Notes in Computer Science*, pages 423–439, Cambridge, USA, 2005. Springer Berlin Heidelberg.

[106] Laurent Bulteau, Guillaume Fertin, Minghui Jiang, and Irena Rusu. Tractability and approximability of maximal strip recovery. *Theoretical Computer Science*, 440:14–28, 2012.

[107] Julien Baste, Christophe Paul, Ignasi Sau, and Celine Scornavacca. Efficient FPT Algorithms for (Strict) Compatibility of Unrooted Phylogenetic Trees. *Bulletin of Mathematical Biology*, 79(4):920–938, feb 2017.

[108] Laurent Bulteau and Mathias Weller. Parameterized Algorithms in Bioinformatics: An Overview. *Algorithms*, 12(12):256, dec 2019.

[109] M. S. Waterman. Secondary Structure of Single Stranded Nucleic Acids. *Advances in Mathematics Supplementary Studies*, 1(1):167–212, 1978.

[110] A. Xayaphoummine, T. Bucher, F. Thalmann, and H. Isambert. Prediction and statistics of pseudoknots in RNA structures using exactly clustered stochastic simulations. *Proc. Natl. Acad. Sci. U. S. A.*, 100(26):15310–15315, 2003.

[111] Tatsuya Akutsu. Dynamic programming algorithms for RNA secondary structure prediction with pseudoknots. *Discrete Appl. Math.*, 104(1-3):45–62, 2000.

[112] R. B. Lyngsø and C. N. S. Pedersen. RNA Pseudoknot Prediction in Energy-Based Models. *Journal of Computational Biology*, 7(3-4):409–427, 2000.

[113] Saad Sheikh, Rolf Backofen, and Yann Ponty. Impact Of The Energy Model On The Complexity Of RNA Folding With Pseudoknots. In Juha Kärkkäinen and Jens Stoye, editors, *CPM - 23rd Annual Symposium on Combinatorial Pattern Matching*, volume 7354 of *Combinatorial Pattern Matching*, pages 321–333, Helsinki, Finland, July 2012. Juha Kärkkäinen, Springer.

[114] Guillaume Blin, Alain Denise, Serge Dulucq, Claire Herrbach, and Helene Touzet. Alignments of RNA Structures. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 7(2):309–322, apr 2010.

[115] Ioanna Kalvari, Eric P Nawrocki, Nancy Ontiveros-Palacios, Joanna Argasinska, Kevin Lamkiewicz, Manja Marz, Sam Griffiths-Jones, Claire Toffano-Nioche, Daniel Gautheret, Zasha

Weinberg, Elena Rivas, Sean R Eddy, Robert D Finn, Alex Bateman, and Anton I Petrov. Rfam 14: expanded coverage of metagenomic, viral and microRNA families. *Nucleic Acids Research*, 49(D1):D192–D200, nov 2020.

[116] Roman Sarrazin-Gendron, Hua-Ting Yao, Vladimir Reinharz, Carlos G. Oliver, Yann Ponty, and Jérôme Waldispühl. Stochastic Sampling of Structural Contexts Improves the Scalability and Accuracy of RNA 3D Module Identification. In *Lecture Notes in Computer Science*, pages 186–201, Padua, Italy, 2020. Springer International Publishing.

[117] Neocles B Leontis and Eric Westhof. Geometric nomenclature and classification of RNA base pairs. *RNA*, 7(4):499–512, 2001.

[118] Vladimir Reinharz, Antoine Soulé, Eric Westhof, Jérôme Waldispühl, and Alain Denise. Mining for recurrent long-range interactions in RNA structures reveals embedded hierarchies in network families. *Nucleic Acids Research*, 46(8):3841–3851, mar 2018.

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

[120] Yinglei Song, Chunmei Liu, Russell Malmberg, Fangfang Pan, and Liming Cai. Tree decomposition based fast search of RNA structures including pseudoknots in genomes. In *Computational Systems Bioinformatics Conference, 2005. Proceedings. 2005 IEEE*, pages 223–234. IEEE, 2005.

[121] Buhm Han, Banu Dost, Vineet Bafna, and Shaojie Zhang. Structural Alignment of Pseudoknotted RNA. *Journal of Computational Biology*, 15(5):489–504, 2008.

[122] Jelena Vucinic, David Simoncini, Manon Ruffini, Sophie Barbe, and Thomas Schiex. Positive multistate protein design. *Bioinformatics*, 36(1):122–130, jun 2019.

[123] Hua-Ting Yao, Jérôme Waldispühl, Yann Ponty, and Sebastian Will. Taming Disruptive Base Pairs to Reconcile Positive and Negative Structural Design of RNA. In *Research in Computational Molecular Biology - 25th international conference on research in computational molecular biology (RECOMB 2021)*, Padova, France, April 2021.

[124] Stefan Hammer, Wei Wang, Sebastian Will, and Yann Ponty. Fixed-parameter tractable sampling for RNA design with multiple target structures. *BMC Bioinformatics*, 20(1), apr 2019.

[125] Ehab S El-Mallah and Charles J Colbourn. The complexity of some edge deletion problems. *IEEE transactions on circuits and systems*, 35(3):354–362, 1988.

[126] Christophe Crespelle, Pål Grønås Drange, Fedor V Fomin, and Petr A Golovach. A survey of parameterized algorithms and the complexity of edge modification. *arXiv preprint arXiv:2001.06867*, 2020.

[127] Leizhen Cai. Parameterized complexity of vertex colouring. *Discrete Applied Mathematics*, 127(3):415–429, 2003.

[128] Marek Cygan, Daniel Lokshtanov, Marcin Pilipczuk, Michał Pilipczuk, and Saket Saurabh. On the hardness of losing width. In *International Symposium on Parameterized and Exact Computation*, pages 159–168. Springer, 2011.

[129] Julien Baste, Ignasi Sau, and Dimitrios M. Thilikos. Hitting Minors on Bounded Treewidth Graphs. I. General Upper Bounds. *SIAM J. Discret. Math.*, 34(3):1623–1648, 2020.

[130] Stefan Arnborg, Jens Lagergren, and Detlef Seese. Easy problems for tree-decomposable graphs. *Journal of Algorithms*, 12(2):308–340, 1991.

[131] Toshiki Saitoh, Ryo Yoshinaka, and Hans L. Bodlaender. Fixed-Treewidth-Efficient Algorithms for Edge-Deletion to Interval Graph Classes. In *Algorithms and Computation - 15th International Conference and Workshops (WALCOM 2021)*, volume 12635 of *Lecture Notes in Computer Science*, pages 142–153, Yangon, Myanmar, 2021. Springer.

[132] Jinsong Tan and Louxin Zhang. The consecutive ones submatrix problem for sparse matrices. *Algorithmica*, 48(3):287–299, 2007.

[133] Andrzej Proskurowski and Jan Arne Telle. Classes of graphs with restricted interval models. *Discrete Mathematics & Theoretical Computer Science*, 3(4), 2006.

[134] Krzysztof Pietrzak. On the parameterized complexity of the fixed alphabet shortest common supersequence and longest common subsequence problems. *Journal of Computer and System Sciences*, 67(4):757–771, 2003.

[135] Michael R Fellows, Danny Hermelin, Frances Rosamond, and Stéphane Vialette. On the parameterized complexity of multiple-interval graph problems. *Theoretical computer science*, 410(1):53–61, 2009.

[136] Hans L Bodlaender. Discovering treewidth. In *International Conference on Current Trends in Theory and Practice of Computer Science*, pages 1–16. Springer, 2005.

[137] Wenzel Jakob, Jason Rhinelander, and Dean Moldovan. pybind11 – Seamless operability between C++11 and Python, 2017. https://github.com/pybind/pybind11.

[138] Helen M Berman, John Westbrook, Zukang Feng, Gary Gilliland, Talapady N Bhat, Helge Weissig, Ilya N Shindyalov, and Philip E Bourne. The protein data bank. *Nucleic acids research*, 28(1):235–242, 2000.

[139] Robert J Klein and Sean R Eddy. RSEARCH: finding homologs of single structured RNA sequences. *BMC bioinformatics*, 4(1):44, 2003.

[140] Elena Rivas and Sean R Eddy. Parameterizing sequence alignment with an explicit evolutionary model. *BMC bioinformatics*, 16(1):406, 2015.

[141] Wei Wang. *Practical sequence-structure alignment of RNAs with pseudoknots*. PhD thesis, Université Paris-Saclay, School of Computer Science, 2017.

[142] Wei Wang, Alain Denise, and Yann Ponty. LicoRNA: aLignment of Complex RNAs v1.0, 2017.

[143] N. Sudarsan, E. R. Lee, Z. Weinberg, R. H. Moy, J. N. Kim, K. H. Link, and R. R. Breaker. Riboswitches in Eubacteria Sense the Second Messenger Cyclic Di-GMP. *Science*, 321(5887):411–413, 2008.

[144] Rita Tamayo. Cyclic diguanylate riboswitches control bacterial pathogenesis mechanisms. *PLOS Pathogens*, 15(2):1–7, 02 2019.

[145] Kathryn D. Smith, Carly A. Shanahan, Emily L. Moore, Aline C. Simon, and Scott A. Strobel. Structural basis of differential ligand recognition by two classes of bis-(3'-5')-cyclic dimeric guanosine monophosphate-binding riboswitches. *Proceedings of the National Academy of Sciences*, 108(19):7757–7762, 2011.

[146] Xiang-Jun Lu, Harmen J. Bussemaker, and Wilma K. Olson. DSSR: an integrated software tool for dissecting the spatial structure of RNA. *Nucleic Acids Research*, 43(21):e142–e142, 07 2015.

[147] J D Thompson, F Plewniak, and O Poch. BAliBASE: a benchmark alignment database for the evaluation of multiple alignment programs. *Bioinformatics*, 15(1):87–88, 01 1999.

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

[149] 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.

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

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

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

[153] Song Cao and Shi-Jie Chen. Predicting RNA pseudoknot folding thermodynamics. *Nucleic Acids Research*, 34(9):2634–2652, 01 2006.

[154] 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.

[155] 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.

[156] 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.

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

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

[159] 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.

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

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

[162] 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.

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

[164] 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.

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

[166] 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.

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

[168] Vincent Bouchitté and Ioan Todinca. Treewidth and minimum fill-in: Grouping the minimal separators. *SIAM Journal on Computing*, 31(1):212–232, 2001.

[169] 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.

[170] J. S. McCaskill. The equilibrium partition function and base pair binding probabilities for RNA secondary structure. *Biopolymers*, 29(6-7):1105–1119, 1990.

[171] Ye Ding and Charles E. Lawrence. A statistical sampling algorithm for RNA secondary structure prediction. *Nucleic Acids Research*, 31(24):7280–7301, 12 2003.

[172] 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.

[173] 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.

[174] 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.

[175] 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.

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

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

[178] 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.

[179] Sean R Eddy and Richard Durbin. RNA sequence analysis using covariance models. *Nucleic acids research*, 22(11):2079–2088, 1994.

[180] Eric P Nawrocki and Sean R Eddy. Query-dependent banding (QDB) for faster RNA similarity searches. *PLoS computational biology*, 3(3):e56, 2007.

[181] Hiroyuki Seki, Takashi Matsumura, Mamoru Fujii, and Tadao Kasami. On multiple context-free grammars. *Theoretical Computer Science*, 88(2):191–229, 1991.

[182] Temple F Smith, Michael S Waterman, et al. Identification of common molecular subsequences. *Journal of molecular biology*, 147(1):195–197, 1981.

[183] Johannes Köster. Rust-Bio: a fast and safe bioinformatics library. *Bioinformatics*, 32(3):444–446, 2016.

[184] Limin Fu, Beifang Niu, Zhengwei Zhu, Sitao Wu, and Weizhong Li. CD-HIT: accelerated for clustering the next-generation sequencing data. *Bioinformatics*, 28(23):3150–3152, 2012.

[185] Paul P Gardner, Andreas Wilm, and Stefan Washietl. A benchmark of multiple sequence alignment programs upon structural RNAs. *Nucleic acids research*, 33(8):2433–2439, 2005.

[186] Andreas Wilm, Indra Mainz, and Gerhard Steger. An enhanced RNA alignment benchmark for sequence alignment programs. *Algorithms for molecular biology*, 1(1):1–11, 2006.

[187] Hua-Ting Yao, Yann Ponty, and Sebastian Will. Developing complex RNA design applications in the Infrared framework, 2022.

[188] Rina Dechter. Reasoning with probabilistic and deterministic graphical models: Exact algorithms. *Synthesis Lectures on Artificial Intelligence and Machine Learning*, 7(3):1–191, 2013.

[189] Hisao Tamaki. A Polynomial Time Algorithm for Bounded Directed Pathwidth. In Petr Kolman and Jan Kratochvíl, editors, *Graph-Theoretic Concepts in Computer Science*, pages 331–342, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

[190] Jan van den Heuvel. The complexity of change. *Surveys in combinatorics*, 409(2013):127–160, 2013.

[191] Daniel Lokshtanov and Amer E Mouawad. The complexity of independent set reconfiguration on bipartite graphs. *ACM Transactions on Algorithms (TALG)*, 15(1):1–19, 2018.

[192] Takehiro Ito, Marcin Kamiński, Hirotaka Ono, Akira Suzuki, Ryuhei Uehara, and Katsuhisa Yamanaka. Parameterized complexity of independent set reconfiguration problems. *Discrete Applied Mathematics*, 283:336–345, 2020.

[193] Rémy Belmonte, Eun Jung Kim, Michael Lampis, Valia Mitsou, Yota Otachi, and Florian Sikora. Token sliding on split graphs. *Theory of Computing Systems*, 65:662–686, 2021.

[194] Amer E Mouawad, Naomi Nishimura, Venkatesh Raman, Narges Simjour, and Akira Suzuki. On the parameterized complexity of reconfiguration problems. *Algorithmica*, 78(1):274–297, 2017.

[195] Daniel Lokshtanov, Amer E Mouawad, Fahad Panolan, MS Ramanujan, and Saket Saurabh. Reconfiguration on sparse graphs. *Journal of Computer and System Sciences*, 95:122–131, 2018.

[196] Daniel Lokshtanov, Amer E Mouawad, Fahad Panolan, and Sebastian Siebertz. On the Parameterized Complexity of Reconfiguration of Connected Dominating Sets. *arXiv preprint arXiv:1910.00581*, 2019.

[197] János Barát. Directed path-width and monotonicity in digraph searching. *Graphs and Combinatorics*, 22(2):161–172, 2006.

[198] Boting Yang and Yi Cao. Digraph searching, directed vertex separation and directed pathwidth. *Discrete Applied Mathematics*, 156(10):1822–1837, 2008.

[199] David Coudert, Dorian Mazauric, and Nicolas Nisse. Experimental evaluation of a branch-and-bound algorithm for computing pathwidth and directed pathwidth. *Journal of Experimental Algorithmics (JEA)*, 21:1–23, 2016.

[200] Joshua Erde. Directed path-decompositions. *SIAM Journal on Discrete Mathematics*, 34(1):415–430, 2020.

[201] Kenta Kitsunai, Yasuaki Kobayashi, Keita Komuro, Hisao Tamaki, and Toshihiro Tano. Computing Directed Pathwidth in $O(1.89^n)$ Time. *Algorithmica*, 75(1):138–157, 2016.

[202] Hans L Bodlaender. Fixed-parameter tractability of treewidth and pathwidth. In *The Multivariate Algorithmic Revolution and Beyond*, pages 196–227. Springer, 2012.

[203] Hisao Tamaki. A directed path-decomposition approach to exactly identifying attractors of Boolean networks. In *2010 10th International Symposium on Communications and Information Technologies*, pages 844–849. IEEE, 2010.

[204] Yasuaki Kobayashi, Keita Komuro, and Hisao Tamaki. Search space reduction through commitments in pathwidth computation: An experimental study. In *International Symposium on Experimental Algorithms*, pages 388–399. Springer, 2014.

[205] Marinus Gottschau, Felix Happach, Marcus Kaiser, and Clara Waldmann. Budget Minimization with Precedence Constraints. *CoRR*, abs/1905.13740, 2019.

[206] Jeff Kinne, Ján Manuch, Akbar Rafiey, and Arash Rafiey. Ordering with precedence constraints and budget minimization. *CoRR*, abs/1507.04885, 2015.

[207] H Abdel-Wahab. On strictly optimal schedules for the cumulative cost-optimal scheduling problem. *Computing*, 24:61–86, 1980.

[208] Evan Senter and Peter Clote. Fast, approximate kinetics of RNA folding. *Journal of Computational Biology*, 22(2):124–144, 2015.

[209] Komei Fukuda and Tomomi Matsui. Finding all the perfect matchings in bipartite graphs. *Applied Mathematics Letters*, 7(1):15–18, 1994.

[210] Zan-Bo Zhang, Xiaoyan Zhang, and Xuelian Wen. Directed Hamilton Cycles in Digraphs and Matching Alternating Hamilton Cycles in Bipartite Graphs. *SIAM J. Discret. Math.*, 27(1):274–289, 2013.

[211] Hisao Tamaki. A polynomial time algorithm for bounded directed pathwidth. In *International Workshop on Graph-Theoretic Concepts in Computer Science*, pages 331–342. Springer, 2011.

[212] Akbar Rafiey, Jeff Kinne, Ján Manuch, and Arash Rafiey. Ordering with precedence constraints and budget minimization. *arXiv preprint arXiv:1507.04885*, 2015.

[213] László Lovász and Michael D Plummer. *Matching theory*, volume 367. American Mathematical Soc., 2009.

[214] Jianer Chen and Iyad A Kanj. Constrained minimum vertex cover in bipartite graphs: complexity and parameterized algorithms. *Journal of Computer and System Sciences*, 67(4):833–847, 2003.

[215] Mark de Berg, Bart MP Jansen, and Debankur Mukherjee. Independent-set reconfiguration thresholds of hereditary graph classes. *Discrete Applied Mathematics*, 250:165–182, 2018.

[216] Rina Dechter. *Constraint processing*. Morgan Kaufmann, 2003.

[217] Gal Elidan and Stephen Gould. Learning bounded treewidth Bayesian networks. *Advances in neural information processing systems*, 21, 2008.

[218] Janne Korhonen and Pekka Parviainen. Exact learning of bounded tree-width Bayesian networks. In *Artificial Intelligence and Statistics*, pages 370–378. PMLR, 2013.

[219] Joost Engelfriet. Context-free graph grammars. In *Handbook of formal languages: volume 3 beyond words*, pages 125–213. Springer, 1997.

[220] Clemens Lautemann. The complexity of graph languages generated by hyperedge replacement. *Acta Informatica*, 27:399–421, 1990.

[221] Hans L Bodlaender. A partial k-arboretum of graphs with bounded treewidth. *Theoretical computer science*, 209(1-2):1–45, 1998.

[222] Édouard Bonnet, Eun Jung Kim, Stéphan Thomassé, and Rémi Watrigant. Twin-width I: tractable FO model checking. *ACM Journal of the ACM (JACM)*, 69(1):1–46, 2021.

[223] Zan-Bo Zhang and Dingjun Lou. Bipartite graphs with a perfect matching and digraphs. *arXiv preprint arXiv:1011.4359*, 2010.

ECOLE DOCTORALE

Titre : Algorithmie paramétrée exacte pour la bioinformatique structurale des ARNs

Mots clés : ARN, agorithmie paramétrée, graphes, largeur arborescente

Résumé : Les ARNs (Acides Ribo-Nucléiques) constituent, avec l'ADN et les protéines, l'un des blocs élémentaires sur lesquels sont construits tous les systèmes biologiques. Si ils sont surtout connus comme étant de simples intermédiaires dans la synthèse de protéines (ARNs messagers), ils peuvent aussi agir directement en tant qu'ARN, et remplir alors des rôles très variés (catalyse, régulation de l'expression de gènes...). Pour ces ARNs dits *non-codants*, la *structure de repliement* qu'ils adoptent est cruciale.

À la fois les séquences et les structures d'ARN présentent un aspect intrinsèquement *combinatoire*: les séquences sont des mots sur l'alphabet `A,U,G,C`, tandis que les structures sont principalement constituées de paires de bases `A-U`, `G-C` et `G-U`. Plusieurs problèmes fondamentaux impliquant les ARNs non-codant sont par conséquent naturellement exprimés dans le language des *mathématiques discrètes*. Ces problèmes incluent le *repliement* (Quelle est la structure préférentielle d'une séquence donnée ?), le *design d'ARN* (Comment trouver une séquence se repliant selon une structure spécifiée en entrée ?) ou le calcul de *barrières d'énergie* (Y'a-t-il une transition entre deux structures susceptible de survenir spontanément ?). Certains de ces problèmes fondamentaux sont NP-difficile, mais ls bioinformaticiens de l'ARN doivent tout de même les résoudre quotidiennement, soit pour mieux comprendre les systèmes biologiques naturels, soit pour le développement de *thérapies à base d'ARN* (dont les vaccins contre le COVID19 sont un exemple). Étant donné également les quantités toujours plus grandes de données de séquençage à traiter, il y a un besoin croissant de méthodes algorithmiques efficaces pour les problèmes mentionnés ci-dessus.

La philosophie de cette thèse de doctorat est d'explorer les possibilités d'application de l'algorithmie *paramétrée*, un domaine relativement récent et très dynamique de la recherche algorithmique, à des problèmes difficiles de bioinformatique des ARNs. Une attention particulière est donnée aux formulations en termes de *graphes*, et à des paramètres de *largeurs de graphes*.

Title : Exact Parameterized Algorithmics for structural RNA bioinformatics

Keywords : RNA, parameterized algorithmics, graphs, treewidth

Abstract :
RNAs are one of the fundamental building blocks of life, along with DNA and proteins. If they are mostly known as a mere intermediate in the synthesis of proteins (*messenger* RNAs), they may also act directly as RNA to perform a wide variety of functions (catalysis, expression regulation...). For these *non-coding* RNAs, the *folded structures* they adopt is crucial.

Both RNA sequences and structures display an inherently *combinatorial* nature: sequences are words over `A,U,G,C`, while structures mainly consist of `A-U`, `G-C` and `G-U` base-pairs. Several fundamental computational problems involving functional RNAs are therefore naturally expressed in the language of discrete mathematics. Such problems include RNA FOLDING (what is the preferred structure of a sequence ?), RNA DESIGN (how do I find a sequence that would fold into a given structure ?) or RNA ENERGY BARRIER (is there a feasible transition between two structures ?). Some of these fundamental problems are NP-hard, but still need to be solved by RNA bioinformaticians in practice, either to better understand biological systems or for the development of RNA therapeutics (e.g. COVID19 vaccines). Combined with the ever-increasing amount of sequencing data available, there is a dire need for efficient methods.

The philosophy of this PhD thesis is to explore the possibility of applying *parameterized algorithmics*, a relatively recent and very dynamic field of algorithmic research, to hard structural RNA bioinformatics problems. A particular focus is given to *graph formulations* and graph *width measures* as parameters.

Institut Polytechnique de Paris
91120 Palaiseau, France