9-2017

A Combinatorial Framework for Multiple RNA Interaction Prediction

Syed Ali Ahmed
The Graduate Center, City University of New York

How does access to this work benefit you? Let us know!
Follow this and additional works at: https://academicworks.cuny.edu/gc_etds
Part of the Bioinformatics Commons, and the Theory and Algorithms Commons

Recommended Citation

This Dissertation is brought to you by CUNY Academic Works. It has been accepted for inclusion in All Dissertations, Theses, and Capstone Projects by an authorized administrator of CUNY Academic Works. For more information, please contact deposit@gc.cuny.edu.
A COMBINATORIAL FRAMEWORK FOR MULTIPLE RNA INTERACTION PREDICTION

by

Syed Ali Ahmed

A dissertation submitted to the Graduate Faculty in Computer Science in partial fulfillment of the requirements for the degree of Doctor of Philosophy, The City University of New York

2017
A Combinatorial Framework for Multiple RNA Interaction Prediction

by

Syed Ali Ahmed

This manuscript has been read and accepted for the Graduate Faculty in Computer Science in satisfaction of the dissertation requirement for the degree of Doctor of Philosophy.

Date

Saad Mneimneh
Chair of Examining Committee

Date

Robert Haralick
Executive Officer

Supervisory Committee

Amotz Bar-Noy, Brooklyn College, City University of New York

Lei Xie, Hunter College, City University of New York

Iman Hajirasouliha, Weill Cornell Medicine

THE CITY UNIVERSITY OF NEW YORK
Abstract

A Combinatorial Framework for Multiple RNA Interaction Prediction

by

Syed Ali Ahmed

Advisor: Saad Mneimneh

The interaction of two RNA molecules involves a complex interplay between folding and binding that warranted recent developments in RNA-RNA interaction algorithms. However, biological mechanisms in which more than two RNAs take part in an interaction also exist.

A typical algorithmic approach to such problems is to find the minimum energy structure. Often the computationally optimal solution does not represent the biologically correct structure of the interaction. In addition, different biological structures may be observed, depending on several factors. Furthermore, scoring techniques often miss critical details about dependencies within different parts of the structure, which typically leads to lower scores (i.e., higher energies). This necessitates development of algorithms to determine not only the optimal solution but also suboptimal solutions, while accounting for dependencies.

We formulate multiple RNA interaction as a combinatorial optimization problem. This problem is NP-hard, so we focus on three aspects in our thesis: 1) design and analyze approximation algorithms for solving the optimization version, 2) develop enumeration and sampling algorithms to generate and cluster suboptimal solutions, and 3) extend existing scoring formulations to account for dependencies within the structure. The inclusion of dependencies in scoring formulations increases the accuracy of sampling algorithms.

For the first task, we develop and implement a dynamic programming based polynomial time approximation scheme. To generate suboptimal solutions, we consider two different approaches. The first is an enumeration algorithm that generates all possible structures within a given fraction
of the optimal energy. The other uses Gibbs sampling and Markov Chain Monte Carlo methods to
draw samples of solutions from the Boltzmann distribution. Since both approaches generate many
structures that may be similar, we develop a distance function to cluster them into unique shapes.
For our third goal, we explore several formulations to capture dependencies, and adopt those that
are computationally tractable. We implement the optimization and approximation algorithms to
predict the optimal solution, as well as the sampling algorithm to predict suboptimal solutions,
and validate their results experimentally.
Acknowledgements

First and foremost, I thank my advisor Saad Mneimneh. I have to thank him for many things. He considered me suitable enough, from a pool of several highly capable candidates, to offer me his mentorship. In the process, he encouraged me to switch to a more applied area than the one I previously had in mind, thus opening up many opportunities for me. Saad’s guidance and support has been instrumental in the completion of this thesis. His students could walk into his office at any time to discuss ideas, which is a privilege that I cherished the most.

I would also like to thank my committee members Amotz Bar-Noy, Lei Xie and Iman Haji-rasouliha for agreeing to serve on my committee and offering many useful suggestions along the way. I also took several seminars with Amotz during my time at CUNY and they helped shape my understanding and appreciation of theoretical computer science. On the other hand, my understanding of the biological aspects of my research would have been lacking without the help of Nancy Greenbaum. I am thankful to her for taking the time to explain biological principles to a computer scientist, and for providing us with data.

I am also indebted to Felisa Vazquez-Abad for making me a part of her lab, CoSSMO, at Hunter College, and the many fruitful discussions that that entailed. Having a permanent base at Hunter College enabled me to foster many relationships with colleagues that were a source of support during my PhD years.

This brings me to all the wonderful friends that I have made at CUNY. I would like to thank Ali Raza Syed, Xing Su, Suman Bhunia, Thomas Flynn, Saman Farhat, Hernisa Kacorri, Anoop Aroor, Allan Zelener and Dara Pir for their support and friendship. A special thanks goes out
to Alexey Nikolaev, for being an amazing labmate, confidant and friend all these years. It goes without saying that this list is not exhaustive.

Many thanks go to Joseph Driscoll at Hunter College for his valuable advice in complicated situations. I would also like to thank the APOs at the Graduate Center, Dilvania Rodriguez and Lina Garcia, for their assistance.

My research was made possible due to funding from many sources, in particular the Science Fellowship from the Graduate Center, a grant from the NSF, a grant from PhRMA, numerous grants from PSC CUNY, and teaching opportunities at Hunter College spanning more than four years.

Finally, everything that I have accomplished is due to the love, motivation and support that I have received from my parents and sister. Their sacrifices have allowed me to pursue my passion. I do not have the words to thank them.
Dedicated to my family
Contents

Abstract iv
Acknowledgements vi
List of Figures xiv
List of Tables xv

1 Introduction 1
1.1 RNA Folding . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 3
1.2 Concatenation: A Model of Multiple RNA Interaction . . . . . . . . . . . . . . 4
1.3 Some Drawbacks of Concatenation . . . . . . . . . . . . . . . . . . . . . . . . . . 4
1.4 Our Model for Multiple RNA Interaction . . . . . . . . . . . . . . . . . . . . . . 6
1.5 Organization of the Thesis . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 7

2 Background 10
2.1 Structure of RNAs . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 10
2.2 Free Energy and the Nearest Neighbor Model . . . . . . . . . . . . . . . . . . . 14
2.3 Canonical Ensemble and Boltzmann Distribution . . . . . . . . . . . . . . . . . 15
2.4 RNA Folding Algorithms . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 16
2.4.1 Nussinov’s Base Pair Maximization Algorithm . . . . . . . . . . . . . . . . 18
2.4.2 Zuker’s Free Energy Minimization Algorithm . . . . . . . . . . . . . . . . 19
2.4.3 McCaskill’s Partition Function . . . . . . . . . . . . . . . . . . . . . . . . . 20
2.5 Advances in Pairwise RNA Interaction Algorithms . . . . . . . . . . . . . . . . 20
2.5.1 A Two Step Process . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 20
2.5.2 Computing Probabilities . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 21

3 Formulation and Optimal Solution 24
3.1 Pegs and Rubber Bands (Base Pair Model) . . . . . . . . . . . . . . . . . . . . . 24
3.1.1 Representing Interactions . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 25
3.1.2 An Algorithm .................................................. 28
3.1.3 Hardness Results .......................................... 30
3.2 Application to Multiple RNA Interaction .................. 32
3.3 Pegs and Rubber Bands (Loop Energy Model) .......... 33
    3.3.1 Weight Function ........................................ 35
    3.3.2 An Algorithm for Windows .......................... 35
3.4 Approximation Algorithms .................................. 36
3.5 Heuristic for Optimal Ordering ............................ 42
3.6 Experimental Results ....................................... 44
    3.6.1 Structure Prediction ................................. 44
    3.6.2 Fishing for Pairs ................................. 46

4 Bipartite Pegs and Rubber Bands ........................ 49
    4.1 Introduction ................................................. 49
    4.2 Heuristic for Optimal Partitioning .................... 52
    4.3 Experimental Results .................................... 52
    4.4 A Special Case: Star Interactions .................... 54

5 Dependent Weights Pegs and Rubber Bands ............... 58
    5.1 Motivation for Dependent Weights .................... 59
        5.1.1 Previous Work ..................................... 60
        5.1.2 Formulation ....................................... 60
    5.2 General Dependencies .................................. 63
    5.3 Nested Dependencies .................................. 65
    5.4 Adjacent Dependencies ................................ 68
        5.4.1 Scoring a Given Structure ..................... 68
        5.4.2 Computing the Optimal Structure .............. 69

6 Suboptimal Solutions I: Enumeration ...................... 74
    6.1 Motivation and Background ............................. 75
    6.2 Generating Structures .................................. 77
    6.3 Distance function ....................................... 81

7 Suboptimal Solutions II: Sampling ......................... 85
    7.1 Introduction ............................................. 85
        7.1.1 Gibbs Sampling ................................. 85
        7.1.2 Metropolis-Hastings ............................ 87
    7.2 Proposal Distribution .................................. 88
7.3 Data Structure ................................................................. 91
7.4 Experimental Results ....................................................... 102
  7.4.1 Structural Variation ................................................... 104
  7.4.2 Artifact Interactions .................................................. 105
  7.4.3 Circular Interactions .................................................. 106
  7.4.4 Star Interactions ....................................................... 106

8 Conclusion ................................................................. 108

References ................................................................. 110
## List of Figures

1.1 Modeling RNA folding. .................................................. 3
1.2 Modeling multiple RNA interaction as folding of concatenated RNA sequences. . 4
1.3 Modeling interaction of three RNAs using concatenation, and the limitations of this approach. .................................................. 6
1.4 Another example of limitations of concatenation ........................................... 7
2.1 Nucleotides, and canonical base pairs adenine-uracil and guanine-cytosine ........ 11
2.2 Various representations of RNA secondary structures, using fhlA as an example . 12
2.3 U6 and U2 sequences, as well as the secondary structure of their interaction complex. .................................................. 13
2.4 Some of the different types of substructures that form in an RNA secondary structure 13
2.5 Loop decomposition and energy contributions .................................................. 15
2.6 The structure of the U6-U2 complex using circular representation. ...................... 17
3.1 Pegs and Rubber Bands .................................................. 25
3.2 Structures with and without cycles .................................................. 26
3.3 A set of two rubber bands containing an overlap, and its mixed graph .............. 27
3.4 An instance of Pegs and Rubber Bands .................................................. 28
3.5 Reduction from $k$-LCS to Pegs and Rubber Bands ........................................... 31
3.6 Multiple RNA interaction within the eukaryotic spliceosome, with U6, U2 and introns .................................................. 32
3.7 A window created by placing rubber bands around pegs

3.8 Adversarial inputs for Algorithms 3.4 and 3.3 to show that they cannot be guaranteed to achieve a better approximation factor for all instances.

3.9 An example of ordering RNAs in Pegs and Rubber Bands

3.10 The structure of the human snRNA complex U6-U2, as predicted by Algorithm 3.4, using a gap size of 4.

3.11 A circular 4-way junction construct of a hairpin ribozyme complex

3.12 Known pairs of interacting RNAs with reasonable solutions.

3.13 The observed structure of CopA-CopT, compared against results from Algorithm 3.2.

4.1 Two examples of interaction graphs, a cycle and a star.

4.2 Three more examples of ribozyme complexes

4.3 Set Cover reduction to Star-PRB

5.1 Two windows which are part of a folding on level $l_1$. The folding is denoted by dotted curves.

5.2 Three pairwise dependent intervals

5.3 A variable gadget and a clause gadget used in the reduction of 3-SAT to General-Dependent-Weights-PRB

5.4 An example of reducing an instance of $k$-LCS to Nested-Dependent-Weights-PRB

5.5 Decomposition of $W(i, j)$ in the optimization algorithm for adjacent dependencies

5.6 Decomposition of $U(k, l)$ and $V(k, l)$

5.7 Decomposition of $P(r, s)$ and $Q(r, s)$

5.8 Expansion of case (f) of $W(i, j)$’s decomposition, as well as decomposition of $T(r, s)$

6.1 U2-U6 snRNA complex obtained by Greenbaum’s lab

6.2 Observed intermolecular base pairs in the CopA-CopT complex

6.3 Three structures with their boundaries and terminal windows
6.4 Structures that differ from each other by a few base pairs  

7.1 Overlaps in adjacent windows  

7.2 Performance of the set based data structure compared to a naive pairwise comparison based approach, during the creation of overlaps and nonoverlaps matrices.  

7.3 A dendrogram visualizing the result of hierarchical clustering  

7.4 The yeast spliceosome with 4 RNAs
List of Tables

4.1 The precision, recall and F1 score of known RNA interaction complexes. Details are discussed separately (including sources for published results). .......................... 53

7.1 Results for the yeast spliceosome. ................................................................. 104
7.2 Results for CopA-CopT, percentage hit followed by rank and F1-score. ........... 106
7.3 Results for the 4-way junction construct in the hairpin ribozyme complex ....... 106
7.4 Results for the hammerhead ribozyme complex, percentage hit followed by rank and F1-score. ................................................................. 107
7.5 Results for the hairpin ribozyme HP-WT complex, percentage hit followed by rank and F1-score ................................................................. 107
7.6 Results for the hairpin ribozyme HP-RJ complex, percentage hit followed by rank and F1-score ................................................................. 107
Chapter 1

Introduction

Ribonucleic acids (RNAs) are important biopolymers that reside inside cells. These nucleic acids are sequences of nucleotides, each of which consists of one of four bases: adenine (A), cytosine (C), guanine (G), and uracil (U). It was once believed that the only role played by RNAs was in the translation of genetic codes to amino acids in proteins. The majority of RNA transcripts, however, do not encode amino acids, and recent developments have shown that these noncoding segments have several important functions outside protein biosynthesis.

RNAs strands also interact with other molecules, including other RNAs strands, to facilitate or regulate certain processes. An example of such an RNA-RNA interaction is RNA interference (RNAi), where a small interfering RNA (siRNA) or a micro RNA (miRNA) can silence genes by targeting their messenger RNAs (mRNA): The siRNA or miRNA binds to the mRNA and triggers a cascade of events that would result in that mRNA being prevented from being translated into a protein.

Some types of RNAs assemble complex molecular machinery such as the spliceosome. In a spliceosome, small nuclear RNAs (snRNA) and protein complexes coordinate to remove introns from pre-mRNA transcripts - a process known as splicing. [BSZ12, SCB04, NG01].

The interaction between RNAs is primarily manifested by base pairing, where two bases bind to each other via hydrogen bonds, creating a base pair. Thus two RNAs interact by forming
base pairs C-G, A-U, and possibly G-U. In fact, RNA folding, a process that has been studied extensively by biologists and computer scientists alike, consists of forming these base pairs within a single RNA. RNA-RNA interaction is a process in which bases on one RNA pair up with bases on another RNA. When only two RNAs are involved, we refer to the process as pairwise RNA interaction. If more than two RNAs interact, we refer to this as multiple RNA interaction. The RNA complex that is a result of the folding or interaction can be defined in terms base pairs, amongst other features. The secondary structure of a complex is some representation of its base pairs.

The secondary structure is chiefly a function of the sequences involved, but not completely. Complexities arise because base pair formation is often influenced by other factors; it is conjectured that the presence of proteins may positively or negatively impact the formation of some base pairs, and sometimes these impacts may be caused by tertiary interactions (not based on hydrogen bonds) between bases. In addition, the geometry of RNA interaction could preclude the formation of some base pairs that may be viable from a computational point of view. In this thesis we make the simplifying assumption that the secondary structure of multiple RNA interaction is completely determined by sequence information. We account for “other factors” by different computational means that will be discussed later.

It is reasonable to ask, if interaction was (ideally) fully deterministic and purely a function of the sequences of RNAs, could we have defined this function? And if it was possible to define this function, was there as way to deal with nondeterminism? Nondeterminism in RNA structure has been addressed in folding and interaction prediction by producing several “suboptimal” secondary structures that are probabilistically likely, or those which are in a specific score range, instead of one computationally optimal structure.

The goal of our work is to develop novel algorithms to predict the secondary structures of multiple interacting RNAs, and we address the problem of nondeterminism in the same way. In the next section, we will introduce RNA folding, and then move on to existing models of multiple RNA interaction which are based on RNA folding.
1.1 RNA Folding

Existing models of multiple RNA interaction are extensions of models used to predict RNA folding, so we will first give an overview of the latter. There are several algorithms to predict RNA folding, but the underlying models are similar. Thus we will deliberately take an abstract approach when describing the model. Note that there are several different choices for the utility function. We could have a simple function that only counts the number of base pairs (thus making RNA folding an exercise in maximizing the number of base pairs). A similar utility function could add weights to the base pairs depending on the bases involved. A more realistic utility function actually depends on the base pairs depending on the bases involved. We discuss these scoring schemes and algorithms in detail in Section 2.4.

As shown in Figure 1.1 (a), the input for RNA folding is a sequence of bases along with candidates for base pairs (in light grey). The goal is to select the subset of these base pairs that maximizes some utility function while satisfying some constraints. The first constraint is that each base participates in at most one base pair. Note that RNAs have directionality; we call one end the 5' end, and the other the 3' end, and it is conventional to number bases in the 5' → 3' direction. That is, the base at the 5' end is numbered 1, the next is 2, and so on; the base on the 3' end is numbered $n$, where $n$ is the length of the RNA. We represent a base pair with the ordered pair $(i, j)$ where $i, j$ refer to the base indices and $i < j$. The second constraint in the standard model of RNA folding is that two base pairs are not permitted to cross. Two base pairs
Concatenation: A Model of Multiple RNA Interaction

1.2 Concatenation: A Model of Multiple RNA Interaction

The generalization of this RNA folding model to multiple RNA interaction is quite straightforward. The RNA sequences are concatenated in some given order, using artificial bases, which results in a new RNA sequence, to which we can now apply RNA folding. In Figure 1.2, we have three RNA sequences that are concatenated by linkers, which are denoted with red colored bases. It is imperative that the RNAs be concatenated in the same order, i.e., the 3' end of one RNA must be followed by the 5' end of the next. After adjusting the scoring scheme slightly to account for intermolecular base pairs, typical RNA folding algorithms can be used to predict the structure of multiple RNAs in a given order. If the number of RNAs is small, we can do this for all permutations to find the global optimal, otherwise some heuristics or stochastic searching methods must be employed.

1.3 Some Drawbacks of Concatenation

The constraint requiring non-crossing base pairs follows from how RNAs typically fold, but occasionally folded RNA structures contain features called “pseudoknots” that translate to such...
1.3. Some Drawbacks of Concatenation

crossing in the standard model. Consider the the structure in Figure 1.1c, which has been reproduced below with annotations.

The two sets of base pairs I and II cross each other.

The approaches to computing the optimal structure now takes an interesting turn. If crossings are permitted, then why do we not use the same model and apply the maximum matching algorithm to predict the structure with the most number of base pairs (or similar, the maximum weights structure), under the same model? The reason for this is that simple maximum matching could result in a set of base pairs that are completely unrealistic. When naturally occurring folded structures have crossings, base pairs are generally stacked together in helices. A helix has base pairs of the form \((i, j), (i + 1, j - 1), (i + 2, j - 2), \ldots, (i + m, j - m)\) where \(m > 0\); thus they have more than one base pair. In the example above, sets I and II are both helices. Once we require that base pairs occur in helices, the problem becomes more complicated when crossings are allowed. Scoring schemes that consider adjacent pairs of base pairs implicitly enforce this requirement. Unfortunately, it has been shown that the general problem of predicting such structures is NP-hard.

When extended to multiple RNAs, crossing base pairs may translate into a feature called kissing loops. Kissing loops form when intramolecular base pairs cross intermolecular base pairs. This can be observed in the following representation of the complex of two RNAs CopA and CopT:
1.4. Our Model for Multiple RNA Interaction

To work around this, perhaps we could make an argument to ignore intramolecular base pairs and focus only on the intermolecular ones, by altering the scoring scheme in some way. However, in the case of multiple RNAs, some interactions are not predicted even in the absence of crossing intramolecular bonds. Figure 1.3 shows a trivial example of three RNAs where RNA 2 interacts with both RNA 1 and RNA 3 in the optimal solution. There are only $(3 - 1)! = 2$ ways to arrange these three RNAs on a circle, and both result in crossing base pairs. Note that reversing or flipping an RNA is not an option since the concatenation model requires all RNAs to have the same direction in the circle, i.e., $5' \rightarrow 3'$.

The above example relies on the fact that RNA 2 has three regions that participate in interactions, since the removal of any interaction results in a non-crossing circular representation. However, even if each RNA has at most two regions that participate in interactions, there may be cases that are not handled by the concatenation approach. Consider the four RNAs in Figure 1.4, where the RNA 4 also interacts with RNA 1. Depending on the permutation of RNAs, the number of crossings may differ. However, the minimum number of crossings is still non-zero, given by permutation $[1, 4, 3, 2]$ in counter-clockwise direction.

1.4 Our Model for Multiple RNA Interaction

In the last decade or so, newer models and algorithms have been developed in the last decade to predict pairwise RNA interaction, to avoid the problems with the concatenation model. Yet it is not immediately obvious how to extend these newer algorithms to handle more than two RNAs.
1.5. Organization of the Thesis

It is reasonable to believe that interactions involving multiple RNAs are generally more complex to be treated pairwise. For example, one might take a greedy approach and choose the optimal pairs of interacting RNAs using existing algorithms. However, as the pair of RNAs is required to fully interact, this will “lock” the interaction pattern of the whole ensemble into a suboptimal state; thus preventing the correct structure from presenting itself as a solution.

In this thesis, we introduce a framework that we call *Pegs and Rubber Bands* that can be used to represent multiple RNA interaction. The framework is considerably different from concatenation approaches and is able to predict several types of structures that cannot be recognized by concatenation based approaches. The framework is easy to visualize; in fact, the “layers” shown on the left in Figures 1.3 and 1.4 are manifestations of Pegs and Rubber Bands, where each RNA is considered as one level, and interactions take place across levels.

1.5 Organization of the Thesis

In Chapter 2, we review some of the biology and physics of RNA folding and interaction, and describe a summary of some RNA folding algorithms. The core component of physics based RNA structure prediction is the concept of the free energy of molecules, and this chapter introduces the energy models used in current approaches. As mentioned earlier, to address nondeterminism our aim is to output several structures. These structures follow a probability distribution related to their energy, which will also be discussed here.

In Chapter 3, we describe the Pegs and Rubber Bands framework in detail. We also present algorithms based on the framework to predict the optimal structure of multiple interacting RNAs,
and show that the problem of predicting the optimal structure is NP-Hard. However, we present a polynomial time approximation scheme. We conclude by validating our algorithms using a small set of known RNA interaction complexes.

In Chapter 4, we relax some constraints and present a generalized version of Pegs and Rubber Bands. We discuss a hardness result for another class of RNA interaction, and the chapter concludes with experimental results of our algorithm on a larger set of known RNA structures.

In Chapter 5, we extend Pegs and Rubber Bands to incorporate dependencies within the RNA structures. Dependencies correspond to the notion that one set of bonds in the structure may influence the existence of bonds in another part of the structure. Dependencies have been explored previously in the literature; however, we describe three different models of dependencies, and show that predicting optimal structures under two of those dependency models is NP-hard, and provide a polynomial time algorithm to compute the optimal structure (for two RNAs) under the third model. Our goal is to incorporate dependencies into our scoring scheme when computing suboptimal solutions via probabilistic sampling. The probability of a structure is a function of its energy, so incorporating dependencies in the computation of the energy of a given structure enhances the accuracy of sampling. Therefore, for our dependency models, we also present methods to more accurately score a given structure by incorporating dependencies.

In Chapter 6, we discuss suboptimal structures. Since the computationally optimal structure is often not the biologically observed one, due to the nondeterminism, we aim to produce several distinct structures. Here we present a recursive algorithm to generate all structures within a given energy range. Many generated structures are similar (for example, there may be a difference of one base pair), which runs contrary to our goal of distinguishing between shapes of RNA complexes. To rectify this, we introduce a distance function that can be used in conjunction with a clustering algorithm to output only those representative structures that are quite different from each other.

In Chapter 7, we present a stochastic sampling method using Markov Chain Monte Carlo methods and Gibbs Sampling to predict suboptimal structures. A data structure that we use in the sampling algorithm is also introduced. Finally, we apply a clustering algorithm along with
the distance function introduced in Chapter 6 to known multiple RNA complexes, and present our experimental results.

We conclude the thesis in Chapter 8.
Chapter 2

Background

2.1 Structure of RNAs

Up till now, we have presented a somewhat simplified picture of the structure of RNAs. Even though we are interested in the bases in an RNA, we left out the detail that nucleic acids are actually chains of nucleotides, which in turn are molecules that consist of a base and a molecule of sugar and phosphoric acid. The bases are all attached in a sequence to a ribose-phosphate backbone. The bases differentiate one nucleotide from another. Because of this, they essentially define the behavior of the nucleotide.

The formation of base pairs is regulated by the geometry of nucleotides. Typical base pairs are guanine-cytosine base pairs and adenine-uracil base pairs. These two types of pairs are known as canonical base pairs, or Watson-Crick base pairs, named after James Watson and Francis Crick who discovered them. It also not entirely unusual to observe uracil-guanine pairs. This pair is known as the Wobble base pair. Very rarely, under specific conditions, we may observe other types of base pairs as well.

Illustratively, the secondary structure may be described as a graph with connections between paired bases on a backbone [DLWP04]. Figure 2.2 shows a few of the different ways in which sec-
2.1. Structure of RNAs

Figure 2.1: Nucleotides and canonical base pairs. On the left, an adenine (A) molecule binds to a uracil (U) molecule via two hydrogen bonds. On the right, a guanine (G) molecule binds to a cytosine (C) molecule via three hydrogen bonds. Source: https://en.wikipedia.org/wiki/File:Base_pair_AT.svg and Base_pair_GC.svg

Secondary structures (of single, folded RNAs) are represented, while Figure 2.3 shows the secondary structure for a complex of two interacting RNAs, U6 and U2.

In these figures, the ends of RNA sequences and secondary structures are labeled with 5′ and 3′. This is because RNAs have directionality, and each RNA sequence has a 5′ end and a 3′ end. Two RNAs interact in opposite directions. In the secondary structure in Figure 2.3, we can say that U6 interacts in the 5′→3′ direction while U2 interacts in 3′→5′.

To refer to specific bases on an RNA $R$, we can assume that the bases are numbered, starting from 1 on the 5′ end to $|R|$ on the 3′ end, where $|R|$ is the length of the RNA. Base pairs can be referred to using the ordered pairs, e.g. $(i, j)$ indicates a base pair between the $i^{th}$ and $j^{th}$ bases. Structural components are defined by an arrangement of paired and unpaired bases. In general, stacking base pairs together increase the stability of a secondary structure, and unpaired bases decrease it. Figure 2.4 depicts the following components that arise in RNA secondary structures:

- **Stacked Pair**: A stacked pair consists of the base pairs $(i, j)$ and $(i + 1, j - 1)$. The two pairs are usually complementary, but may contain non canonical base pairs as well. A stacked pair increases the stability of a structure.
2.1. Structure of RNAs

5’ AGUUAGUCAAUGACCCCUUUGCCACCCGCUUUGCGGUGCUUCCUUGGAAGAAACAAAAUGCG
UCAUAUACACCCGAGUGAGAUCUCGGACAACCAAGGGGUUGUUCGACAUCACUGGACA 3’

Figure 2.2: Various representations of RNA secondary structures, using fhlA as an example. From top to bottom: the actual fhlA Sequence; outer planar graph (derived from RNAfold), circle plot (drawn by our tool), dot-bracket notation (balanced parentheses expression - generated by RNAfold).

- **Stem**: A stem or helix is a sequence of consecutive stacked base pairs, without having any unpaired base in this segment. Thus a stem consists of base pairs \((i, j), (i+1, j-1), (i+2, j-2)\), and so on.

- **Hairpin Loop**: A hairpin loop is the interval \([i, j]\) is the enclosed by the base pair \((i, j)\) such that bases \(i+1, \ldots, j-1\) do not form any base pairs.

- **Pseudoknot**: Pseudoknots occur when an RNA has non-nested or crossing base pairs, i.e, for two base pairs \((i, j)\) and \((k, l)\), \(i < k < j < l\).

- **Kissing Loop**: A kissing loop is a component of an interaction structure. It is the entire substructure that consists of hairpin loops on both RNAs as well as the interaction between the unpaired intervals of one hairpin with that of the other.

These structural components are actually based on loops. Loops are parts of the RNA structure enclosed by contiguous base pairs. For example, a stacked pair is consists of two contiguous base pairs, so a stacked pair is one loop. A stem or helix is a collection of loops, since
2.1. Structure of RNAs

U6: 5’ GGCCGAUACAGAAGAUUAGCAUGGCCUCUGCAUAAGGAUGACACGCAAAUUCGUGAAGCGUCC 3’

U2: 5’ GGACGCUUCACGGCCUUUGCUAAGUCAAGUGUAGUAGUAUCGGCC 3’

Figure 2.3: U6 and U2 sequences, as well as the secondary structure of their interaction complex.

Figure 2.4: Some of the different types of substructures that form in an RNA secondary structure. The grey lines indicate the RNA backbone, and black lines indicate base pairs. For clarity, we will often omit drawing the RNA backbone.

It is a collection of stacked pairs. The hairpin loop is a special case, since it is enclosed by only one base pair. Every secondary structure can be decomposed into loops in a unique way.
2.2 Free Energy and the Nearest Neighbor Model

The earliest algorithms in RNA structure prediction (e.g., Nussinov’s folding algorithm [NJ80]) were basepair maximization algorithms; i.e., they produced the structure with the maximum number of basepairs (or with maximum weight, in the case of weighted base pairs). However, RNA structure formation is guided by thermodynamics, and the thermodynamic stability of an RNA structure is governed not by the individual base pairs but rather by loops. Therefore, maximizing the number or total weight of base pairs only roughly estimates the structure. Instead, any algorithm that correctly predicts either folding or interaction structures must consider the Gibbs free energy [Gib57] of loops. The stability of an RNA structure is measured in terms of the change in the Gibbs free energy of the solution, which is denoted $\Delta G$. For a loop, $\Delta G$ specifically refers to the loss or gain in energy in when that particular loop is formed, since loop formation is a chemical reaction. It is actually defined in terms of the increase or decrease in entropy and enthalpy: $\Delta G = \Delta H - T \Delta S$, where $\Delta H$ is the change in the enthalpy of the system (in joules), $T$ is the temperature (in Kelvin) and $\Delta S$ is the change in the entropy (in joules/kelvin). Henceforth we will refer to $\Delta G$ simply as the free energy.

Each loop in an RNA structure makes a contribution to the free energy $\Delta G(S)$ of the entire structure $S$. According to commonly used models, stacking pairs and and a few other loops have $\Delta G < 0$ and stabilize the structure, whereas most other loops have $\Delta G > 0$. These contributions are additive, and the free energy of the structure $S$ is the sum of the free energies of its loops.

Figure 2.5 from [DEKM98] shows how the components of the (hypothetical) structure contributes a positive or negative energy. The sum of the free energies of all components is -4.6 kcal/mol. The free energies used in the figure are from the Freier nearest neighbor model, which was one of the first ever compilation of such energies.

In the nearest neighbor model the contribution of a loop depends not only on the type of loop but also on the bases involved in the loop and their neighbors. These values were measured by Tinoco et al in 1971 [TUL71], Freier et al in 1986 [FKJ+86] and by Turner and Matthews in 1987, 1999 and 2004 [TSJ+87, MSZT99, MDC+04], and the model is known as the
2.3. Canonical Ensemble and Boltzmann Distribution

Figure 2.5: Loop decomposition and energy contributions, energy according to Freier rules [DEKM98].

nearest neighbor model. Physics based models for structure prediction use these values when considering the different types of loops that could appear in a subsequence of RNA. The free energies of structures and their components depend on the temperature of the system. Values have been recorded for $37^\circ$ and a few other temperatures; others can be interpolated.

Vis-a-vis RNA folding and interaction, a favorable chemical process is one where $\Delta G < 0$, and therefore predicting the computationally optimal structure becomes an exercise in minimizing free energy. Theoretically, the correct structure is the one with the lowest free energy. However, the structure of an RNA may depend on other factors as well, so this may not always be true. These may include structural features such as coaxial stacking, or the presence of proteins. Often, the structure may be caught in a kinetic trap during the folding process, where it is required to gain free energy before moving to a more favorable, lower energy state [FHSW02, TRZW98]. Such structures are known as locally optimal. Therefore, algorithms that predict a range of structures with varying energy are more useful.

2.3 Canonical Ensemble and Boltzmann Distribution

If the goal of prediction algorithms is to determine which structures are more likely to be observed, instead of one computationally optimal structure, they must consider the distribution of all possible structures (known as states in statistical mechanics) in the canonical ensemble. This
can be used to compute highly probable structures or features. The probability of a structure is related to its energy, i.e., the probability of observing structure $S_i$ in equilibrium is

$$P(S_i) \propto e^{-\frac{\Delta G(S_i)}{RT}}$$ (2.1)

where $R = 0.001987 \text{kcal/mol/K}$ is the universal gas constant and $T$ is the temperature. To compute the probability, we need a normalization constant, which is called the partition function $Z$, i.e.,

$$Z = \sum_{S_i \in S} e^{-\frac{\Delta G(S_i)}{RT}}$$ (2.2)

where $S$ is the set of all possible structures. Therefore, a state $i$ in the Boltzmann distribution has the probability $e^{-\Delta G(S_i)/RT}/Z$.

Eqs. 2.1 and 2.2 give us some information about the ensemble. The lower the energy, the higher the probability of observing that state; hence the most probable structure is the MFE structure. The probability of observing individual features can also be computed similarly. Examples of individual features that are often of interest include particular base pairs, or free regions, i.e., no bases in some range of bases make base pairs with any other bases on the RNA. In order to compute the probability of a feature, we first compute the $Z' = \sum_{S_i \in S'} e^{-\frac{\Delta G(S_i)}{RT}}$ where $S'$ is the set of of structures that contain that feature, and then normalize it by dividing by $Z$.

2.4 RNA Folding Algorithms

RNA secondary structure prediction research dates back to the 1971, with the first algorithm developed by Tinocho et al [TUL71]. Since then, a host of algorithms have been developed, which can also predict, to some extent, non standard features such as pseudoknots. Often, information about the entire set of possible configurations and their probabilities is required in terms of the Boltzmann weight and partition function, and methods to compute the partition function
2.4. RNA Folding Algorithms

Figure 2.6: The structure of the U6-U2 complex using circular representation.

efficiently have also been developed. Most algorithms can be classified in one of three categories:
1) Free energy minimization, 2) algorithms that compute the partition function or make use of
it, and 3) knowledge based structure prediction algorithms, e.g. those that use stochastic context
free grammars (SCFGs).

The first two categories are physics based models, in that the hyperparameters used in the
computation were experimentally determined in vitro (i.e., the Nearest Neighbor model). As
discussed earlier, more accurate models assume RNA free energy is the sum of all loops free ener-
gies, and contemporary physics based tools are all loop based. Popular RNA structure prediction
tools are $\text{Mfold}$, $\text{RNAfold}$ and $\text{RNAstructure}$; the MFE prediction is based on the Zuker algo-
rithm [ZS81] and the partition function computation is by way of McCaskill’s algorithm [McC90].
For a given RNA, these algorithms run in $O(n^3)$ time with $O(n^2)$ space complexity, where $n$
is the length of the RNA. $\text{RNAfold}$ is part of the ViennaRNA suite [LBZS11] which contains many
other tools for RNA analysis, including $\text{RNAup}$ which computes RNA interaction regions for a
pair of RNAs, $\text{RNAsubopt}$ [WFH+99] which computes a number of structures energetically close
to the MFE structure, $\text{RNAeval}$ to compute the energy of a given structure over a given sequence,
and $\text{RNAcofold}$ which uses a concatenation approach to predict the joint structure of two RNAs.
A statistical algorithm to draw suboptimal structures from the canonical ensemble was developed
by Ding and Lawrence and implemented in the program $\text{Sfold}$ [DCL04].
A limitation of most models is that they cannot predict pseudoknots. The prediction of pseudoknots has been proven to be an NP-complete problem [LP00]. However, it is possible to predict a restricted set of pseudoknots using polynomial time deterministic algorithms and by sampling methods. Deterministic algorithms have been developed by Rivas et al [RE99], Lyngsø et al [LP00], Dirks et al [DP03], Chen et al [CCJ09], and others, while a sampling algorithm based on a simple SCFG was developed by Metzler et al [MN08]. Deterministic algorithms run in at least $O(n^4)$ time, so there is no known algorithm that matches the time complexity of psuedoknot-free folding algorithms.

To motivate the dynamic programming approach used by most algorithms, we first discuss the Nussinov base pair maximization algorithm [NJ80], which although not biologically accurate, still forms the basis of all dynamic programming approaches. The Nussinov algorithm seeks to maximize the number of non-crossing base pairs. After that we present Zuker’s algorithm for MFE, followed by McCaskill’s algorithm for the partition function.

2.4.1 Nussinov’s Base Pair Maximization Algorithm

Simply maximizing the number of base pairs in an RNA reduces to finding a maximum matching in the secondary structure graph. However, this allows non-crossing edges. Now one may argue that this may actually be a benefit because it can predict pseudoknots. However, unguided addition of non crossing base pairs leads to structures that are biologically meaningless and have “pseudoknots” that are unrealistic.

To prevent crossing base pairs, Nussinov et al [NJ80] developed a dynamic programming algorithm that, in principle, recursively computes the best structure of a given subsequence based on smaller sequences. The recursive formulation is described below. The score - maximum number of non-crossing base pairs - of the subsequence $i \ldots j$ is stored in a $O(n^2)$ matrix $N$, where $n$ is the length of the RNA.
2.4. RNA Folding Algorithms

\[
N(i, j) = \max \begin{cases} 
N(i + 1, j - 1) + w(i, j) & \text{(a)} \\
\max_{i < k < j} N(i, k) + N(k + 1, j) & \text{(bifurcation)} \\
N(i + 1, j) & \text{(c)} \\
N(i, j - 1) & \text{(d)} 
\end{cases}
\]

\[
N(i, i) = 0 \quad \forall \quad 1 \leq i \leq n
\]

where

\[
w(i, j) = \begin{cases} 
1 & \text{if } i \text{ and } j \text{ are complementary} \\
0 & \text{otherwise}
\end{cases}
\]

Case (a) of the recursion corresponds to inserting a base pair between \( i \) and \( j \) if the bases are complementary, while case (b) corresponds to splitting the problem into two separate subproblems. The time and space complexities of this algorithm are \( O(n^3) \) and \( O(n^2) \) respectively, owing to filling out \( O(n^2) \) entries, with a worst case time of \( O(n) \) (in case 2) for each entry.

The Nussinov algorithm has a very naive scoring scheme, for at least two reasons: 1) the scoring function \( w(.) \) is binary, 2) the algorithm does not take into account neighboring arcs (and therefore stacking energies and internal loops). The scoring function could be modified, so that weights are derived from the distribution of different types of base pairs in large datasets, or so that the score of a base pair depends on the strength of its hydrogen bonds. However, it does not take into account the energy contribution of loops. The next algorithms we discuss are based on the loop-energy model, and do not suffer from these problems.

2.4.2 Zuker’s Free Energy Minimization Algorithm

For a given RNA \( R \), Zuker’s algorithm \([ZS81]\) finds one optimal secondary structure, i.e., the secondary structure with lowest free energy:

\[
mfe(R) = \min_{S \in S} E(S)
\]
where \( S \) is the set of all possible structures of RNA \( R \). Zuker’s algorithm and others do not explicitly consider each structure over \( R \). Instead, they use dynamic programming to cover all cases by finding the best substructure for all subsequences of \( R \), and by considering all types of loops that can be formed in that subsequence. Zuker’s algorithm has the major components of Nussinov’s, specially with respect to the bifurcation technique. However, since Zuker’s algorithm computes loop energies, the computation involves more bookkeeping.

### 2.4.3 McCaskill’s Partition Function

As discussed in Section 2.3, we need to compute the partition function to determine probabilities of states and substructures. The partition function \( Z \) of RNA sequence \( R \) is defined by

\[
Z = \sum_{S \in S} e^{-[E(S)/RT]} \hspace{1cm} (2.7)
\]

\[
= \sum_{S \in S} \prod_{L \in S} e^{-[E(L)/RT]} \hspace{1cm} (2.8)
\]

where \( S \) is the set of all possible structures of RNA \( R \), and any structure \( S \) can be decomposed into a set of loops \( \{L\} \). The number of all possible structures grows exponentially with the sequence length, hence a more efficient method has to be applied. McCaskill’s algorithm [McC90] provides an efficient method to compute the partition function.

### 2.5 Advances in Pairwise RNA Interaction Algorithms

In this section we will describe some recent methods to predict the structures of two interacting RNAs. These methods will later form the basis of the scoring function in our Pegs and Rubber Bands framework.

#### 2.5.1 A Two Step Process

Interaction is now thought of as a two step process. If regions on two RNA strands form intermolecular bonds, they first have to become unpaired. i.e., eliminate any intramolecular bonds
that are present in those regions. After the regions become free, they may form intermolecular bonds. When computing probable interaction structures, a technique that has proved successful is to first determine the regions on each RNA that are likely to be unpaired during folding. This likelihood of the region being free is related to the energy required to eliminate the intramolecular bonds, as we show later. The next step is to compute the strength of the intermolecular bonds, then determine whether the energy contribution of interaction outweighs the energy required to free those regions.

2.5.2 Computing Probabilities

Suppose we wish to determine whether region \([i,j]\) on an RNA \(R\) interacts with a region on another RNA. To determine its probability of being unpaired, we have to consider all possible structures of \(R\) in which \([i,j]\) is unpaired. We can compute a restricted partition function over the set of all structures where \([i,j]\) is unpaired, and call it \(Z_{R(i,j)}^U\). The required probability can be computed using the following equation:

\[
P_R(\text{free}[i,j]) = \frac{Z_{R(i,j)}^U}{Z_R}
\]

(2.9)

where \(Z_R\) is the partition function of RNA \(R\).

There is another way to derive this quantity in terms of energies. Define the ensemble energy of structures with \([i,j]\) unpaired as \(E_R^{U(i,j)} = -RT \ln(Z_{R(i,j)}^U)\), and the ensemble energy of all structures as \(E_R = -RT \ln(Z_R)\). Then the energy required to make the region \([i,j]\) accessible is

\[
E_R(\text{free}[i,j]) = E_R^{U(i,j)} - E_R
\]

(2.10)

Thus \(E_R(\text{free}[i,j]) = -RT \ln P_R(\text{free}[i,j])\), or equivalently, \(P_R(\text{free}[i,j]) = \exp[-E_R(\text{free}[i,j])/RT]\). These quantities are computed for all possible regions \([i,j]\) \(\in R\).

However, to determine whether it is feasible to pair a region \([i,j]\) on RNA \(R\) with a region \([k,l]\) on RNA \(S\), the energy of interaction of these regions needs to be computed. Given the
aforementioned two step process, it can represented with the following equation:

\[
\Delta E = E_R(\text{free}[i,j]) + E_S(\text{free}[k,l]) + E_{RS}^I[i,j,k,l] \tag{2.11}
\]

where \(E_{RS}^I[i,j,k,l]\) is the energy of interaction between two regions on \(R\) and \(S\), which can also be thought of as the gain in energy due to hybridization. If \(\Delta E < 0\) (recall negative energy values indicate favorable reactions) then interaction is feasible.

Efficiently computing \(P_R(\text{free}[i,j])\) is not as straightforward as implied by eq (2.9). In RNAup [MTH+06] a restricted partition function is used with two cases - either \([i,j]\) is not covered by a base pair, or it is. In the former case, the partition function is easy to compute, since it is just the product of partition functions on either side of \([i,j]\) (the partition function of the substring \([i,j]\) is 1). Computing the latter case requires considering all types of loops that may be involved: \([i,j]\) could be on a hairpin loop's unpaired region, it could be on an interior loop, or it may a part of a multiloop. Assuming that \([i,j]\) is covered by a bond between \(p\) and \(q\) in the second case, we have:

\[
P_R(\text{free}[i,j]) = \frac{Z[l,i-1] \times 1 \times Z[j+1,N]}{Z} + \sum_{p \leq i < q} P_{pq} \times \frac{Z_{pq}[i,j]}{Z^b[p,q]} \tag{2.12}
\]

In the summation, we compute the conditional probability that \([i,j]\) is unpaired given that \(p\) and \(q\) form an arc, hence the division by \(Z^b[p,q]\), which is the partition function of all structures in which \(p\) and \(q\) form an arc. For a breakdown of \(Z_{pq}[i,j]\) into cases involving loops and implementation details, see [MTH+06]. This has a running time of \(O(n^4)\), however, in reality long unpaired regions are rare, and if their length is bounded by \(w\), the running time scales down to \(O(n^3 w)\).

The above idea is employed in RNAup by Muckstein et al [MTH+06] and in IntaRNA [BRB08], although they differ in how the final quantities are computed. Muckstein et al consider an ensemble of simple structures for the interaction partition function. They consider only interior
2.5. Advances in Pairwise RNA Interaction Algorithms

loops formed between the two RNA strands, and thus have:

\[
Z^I[i, j, i^*, j^*] = \sum_{i < k < j \atop i^* > k^* > j^*} Z^I[i, k, i^*, k^*] \exp^{-I(k, k^*; j, j^*)}
\] (2.13)

Thus the partition function of all structures where \([i, j] \in R\) binds to \([k, l] \in S\) is given by

\[
Z^*[i, j, k, l] = P_R^{U(i, j)} \times P_S^{U(i, j)} \times Z^I[i, j, k, l]
\] (2.14)

Taking the log on both sides is equivalent to eq (2.11). The quantity \(-RT \ln Z^*[i, j, k, l]\) is ensemble energy of interaction between \([i, j]\) on \(R\) and \([k, l]\) on \(S\). With some modifications, we will incorporate this idea into Pegs and Rubber Bands to model multiple RNA interaction.
Chapter 3

Formulation and Optimal Solution

In this chapter, we describe in detail the framework we use to represent multiple RNA interaction, along with algorithms that predict the computationally optimal structure. We advocate a combinatorial optimization framework that we call Pegs and Rubber Bands to represent multiple RNA interaction. In this representation, each part of an RNA contributes certain a weight to the entire interaction when binding to different parts of other RNAs and we, therefore, seek to maximize the total weight. This notion of weight can be obtained by using existing RNA-RNA interaction algorithms on pairs of RNAs. We show that under certain restrictions, which are similar to those that prevent pseudoknots, the problem remains \textbf{NP}-hard.

We begin by describing Pegs and Rubber bands, and define an optimization problem on it. After showing that the problem is \textbf{NP}-hard, we present an exponential time optimization algorithm. Later we describe a polynomial time approximation scheme (PTAS) for the problem. The chapter concludes with a discussion of the results we obtain on different known instances of multiple RNA interaction.

3.1 Pegs and Rubber Bands (Base Pair Model)

The Pegs and Rubber Bands framework consists of $m$ levels numbered 1 to $m$ with $n_l$ pegs in level $l$ numbered 1 to $n_l$. A rubber band can be placed around two pegs in adjacent levels, i.e.,
3.1. Pegs and Rubber Bands (Base Pair Model)

Figure 3.1: Pegs and Rubber Bands: $m$ levels, numbered 1 to $m$. Each level has pegs numbered 1 to $n_l$. Rubber bands can be placed around pegs in adjacent levels.

in levels $l$ and $l + 1$. For instance, we may choose to place a rubber band around peg $i$ in level $l$, (call this peg $(l, i)$), and peg $j$ in level $l + 1$, (call this peg $(l + 1, j)$); we call it a rubber band at $(l, i, j)$. Each rubber band has a weight, and in this case the rubber band contributes a given weight $w(l, i, j)$. That is, the weight of a rubber band is entirely dependent on the pegs it is wrapped around. Optionally, we may allow levels $m$ and 1 to be adjacent as well, thus allowing a rubber band to wrap around the levels. In this case, the weight $w(m, i, j)$ corresponds to a rubber band band wrapped around peg $i$ in level $m$ and peg $j$ in level 1.

Pegs and Rubber Bands can represent multiple RNA interaction in the following way. RNA sequences become the levels, the ordered pegs in each level represent RNA bases \{A,C,G,U\} in the order of occurrence in their sequence, a rubber band around two pegs is a potential interaction between the corresponding base pair, and the weights are a function of the energies of interacting bases (as base pairs). Since a base $i$ can create at most one base pair (either on the same RNA or a different one), we would ideally want $w(l, i, j)$ to consider both intermolecular and intramolecular base pairs. For example, if base $i$ of RNA $l$ is more likely to create an intramolecular base pair than to bind to base $j$ on RNA $l + 1$, then we may set $w(l, i, j)$ to a negative value. This avoids having to separately handle folding within RNAs.

3.1.1 Representing Interactions

The Pegs and Rubber Bands framework captures an instance of the multiple RNA interaction problem. We now introduce a way to represent a valid solution. Define a structure as a set of
3.1. Pegs and Rubber Bands (Base Pair Model)

(a) In this representation, it appears as if the structure has overlapping rubber bands...
(b) ... but shifting the level along the horizontal axis fixes the “problem”.
(c) An actual overlap between two rubber bands in different levels that cannot be removed by translation or scaling.

Figure 3.2: Structures with and without cycles. The first two are equivalent and cycle-free. The third has a cycle.

rubber bands wrapped around pegs such that there are no overlaps. For a set of rubber bands to not have an overlap, we require two conditions to be true. Firstly, two different rubber bands cannot be attached to the same peg. Secondly, rubber bands may not “cross” each other. For the case of rubber bands between the same levels, the definition of a crossing is intuitive. Suppose we have two rubber bands placed at \((l, i_1, j_1)\) and \((l, i_2, j_2)\). Then there is no overlap if and only if \(i_1 < i_2 \iff j_1 < j_2\). Figure 3.3 shows an example of how such an overlap occurs.

Since it is possible to wrap a rubber band between pegs in the first and last levels as well, the above definition of overlaps does not generalize, i.e., it is not sufficient to consider only those rubber bands which are between the same levels. For example, consider Figure 3.2a, which shows a set in which a rubber band between levels 1 and 4 is drawn over a rubber band between levels 2 and 3. Suppose we are allowed to transform any pictorial representation by 1) shifting the levels to the left or the right along the horizontal axis, and 2) stretching or shrinking any level, so that the distance between some pegs increases or decreases. Then this structure can be redrawn into Figure 3.2b by simply shifting level 4 to the right. The new representation does not have rubber bands that intersect, and thus does not really have crossings. On the other hand, the rubber bands in Figure 3.2c cannot be transformed into a crossing-free representation, and therefore the set truly has a crossing between rubber bands.

To formalize this definition of overlaps, we interpret a set of rubber bands as a mixed graph.
Definition 3.1. A mixed graph $MG = (V, E, A)$ consists of a set of vertices $V$, a set of undirected edges $E$, and a set of directed edges $A$ [BBC+15].

For a given set of rubber bands, we derive the mixed graph from the pegs and rubber bands model using the following transformations:

- Pegs become the set of vertices $V$,
- Rubber bands become undirected edges $E$,
- Each ordered pair of “covered” pegs on the same level becomes a directed edge $A$.

A covered peg is one which has a rubber band attached to it. In the example in Figure 3.3, the mixed graph would be given by $V = \{i_1, i_2, j_1, j_2\}$, $E = \{(i_1, j_2), (i_2, j_1)\}$ and $A = \{(i_1, i_2), (j_1, j_2)\}$. Note that this graph contains the cycle $i_1 \rightarrow i_2 \rightarrow j_1 \rightarrow j_2 \rightarrow i_1$. With this, we can give a general definition of overlaps:

Definition 3.2. A set of rubber bands $S$ contains an overlap if a peg has more than one rubber bands wrapped around it or its mixed graph $MG(S)$ contains a cycle.

We define the weight of a structure as the sum of the weights of all the rubber bands in the structure:

$$w(S) = \sum_{(l,i,j) \in S} w(l, i, j)$$

We can now formally define the Pegs and Rubber Band Problem.

Definition 3.3. The Pegs and Rubber Bands problem (PRB) is to maximize the total weight by placing rubber bands around pegs in adjacent levels in such a way that the optimal structure contains no overlaps.
Figure 3.4 shows an example of Pegs and Rubber Bands with 6 levels, where each available rubber band has unit weight. The figure on the left shows the set of available rubber bands as dashed lines, and the solution to the Pegs and Rubber Band Problem (i.e., the structure with the maximum weight) is show on the right.

3.1.2 An Algorithm

The Pegs and Rubber Bands problem described above can be solved by the recursive formulation in Algorithm 3.1. Let $T(i_1, i_2, \ldots, i_m)$ be the optimal weight when we truncate the levels at pegs $(1, i_1), (2, i_2), \ldots, (m, i_m)$. Then $T(n_1, n_2, \ldots, n_m)$ is the optimal weight.

If $n = \max n_l$, the running time of the algorithm is $O(mn^m)$, which is exponential. Later in this section, we will present a polynomial time approximation scheme. To motivate the rationale behind Algorithm 3.1, we first consider the case of only two levels, for which the recurrence reduces to:

$$T(i, j) = \max \begin{cases} 
T(i - 1, j - 1) + w(1, i, j), \\
T(i, j - 1), T(i - 1, j) 
\end{cases} \quad (3.1)$$

Let $\text{OPT}_{i,j}$ be the optimal solution up to and including peg $i$ on level 1 and peg $j$ on level 2. We use $\text{OPT}_{i,j}$ to denote both the set of edges in, and the value of, the optimal solution of the subproblem involving pegs $1, \ldots, i$ of level 1 and pegs $1, \ldots, j$ of level 2. We observe that there

![Figure 3.4: An instance of Pegs and Rubber Bands. Here, all positive weights are equal to 1, and are represented by dashed lines (left). The optimal solution (right) achieves a total weight of 8.](image)
Algorithm 3.1: Pegs and Rubber Bands Optimization Algorithm

**Inputs**: $m$ levels

**Output**: Optimal weight

$$T(i_1, i_2, \ldots, i_m) = \max \begin{cases} 
T(i_1 - 1, i_2, \ldots, i_m) \\
T(i_1, i_2 - 1, i_3, \ldots, i_m) \\
\vdots \\
T(i_1, \ldots, i_{m-1}, i_m - 1) \\
T(i_1 - 1, i_2 - 1, i_3, \ldots, i_m) + w(1, i_1, i_2) \\
T(i_1, i_2 - 1, i_3 - 1, \ldots, i_m) + w(2, i_2, i_3) \\
\vdots \\
T(i_1, \ldots, i_{m-2}, i_{m-1} - 1, i_m - 1) + w(m - 1, i_{m-1}, i_m) \\
T(i_1 - 1, \ldots, i_{m-2}, i_{m-1}, i_m - 1) + w(m, i_m, i_1) \text{ (wrap around)} 
\end{cases}$$

where $w(l, i, j) \in \mathbb{R}$, and $T(0, 0, \ldots, 0) = 0$.

are four possible scenarios involving pegs $i$ and $j$ (on levels 1 and 2 respectively) in the optimal solution $\text{OPT}_{i,j}$:

1. **There is a rubber band between $i$ and $j$.**
   In this scenario, $\text{OPT}_{i-1,j-1} = \text{OPT}_{i,j} - w(1, i, j)$.

2. **There is a rubber band between $i$ and $j'$, where $j' < j$ (therefore $j$ remains unpaired).**
   There is no peg between $j'$ and $j$ that has a rubber band with any peg on level 1, since such a rubber band will violate the non-overlap condition of the framework. In other words, no rubber band $(k, l) \in [1, i] \times [1, j]$ can be added to $\text{OPT}_{i,j'}$. Thus $\text{OPT}_{i,j} = \text{OPT}_{i,j'} = \text{OPT}_{i,j-1}$.

3. **There is a rubber band between $i'$ and $j$, where $i' < i$ (therefore $i$ remains unpaired).**
   The argument is symmetric to that of scenario 2.

4. **Both $i$ and $j$ remain unpaired.**
   In this scenario, $\text{OPT}_{i,j} = \text{OPT}_{i-1,j-1}$. 
The first case of the recurrence in Eq. 3.1 covers scenario 1, while cases 2 and 3 cover scenarios 2 and 3, respectively. Scenario 4 is covered by an application of case 2 and then case 3 (or vice versa).

We observe that the cases discussed above correspond to whether pegs $i$ and $j$ form pairs (either with each other or with some other pegs) or remain unpaired. We can extend this formulation to all levels and pairs of levels in Algorithm 3.1. That is, when computing $T(i_1, i_2, \ldots, i_m)$ the cases generalize to a) whether peg $i_k$ is unpaired, or b) whether it forms a pair with another RNA. For peg $i_k$ (on level $k$), it is sufficient to test only rubber bands $(k-1, i_{k-1}, i_k)$ and $(k, i_k, i_{k+1})$, since it can only form pairs with pegs on levels $i_{k-1}$ and $i_{k+1}$.

### 3.1.3 Hardness Results

We show that the Pegs and Rubber Bands problem is NP-hard by a reduction from the $k$-Longest Common Subsequence Problem ($k$-LCS).

**Definition 3.4.** ($k$-Longest Common Subsequence Problem) In the $k$-LCS problem, the input consists of $k$ strings $S_1, S_2, \ldots, S_k$ from the alphabet $\Sigma$ and the goal is to find the length of the longest string $R$ such that $R$ is a subsequence of all string $S_1, S_2, \ldots, S_k$.

Note that a subsequence of string $S$ contains characters in the same order as $S$ but are not necessarily contiguous. The decision version of the problem has another input $L$, and asks if there is a common subsequence $R$ such $|R| \geq L$. It has been shown that $k$-LCS is NP-hard [Mai78, JL95]. $k$-LCS is also NP-hard when all strings are binary, i.e., $\Sigma = \{0, 1\}$.

**Theorem 3.1.** Pegs and Rubber Bands is NP-hard.

**Proof.** We make a reduction from the binary $k$-LCS, where the length of the $i^{th}$ string is $n_i$. In this reduction, pegs are labeled with a symbol from $\{0, 1, b, \ast\}$ and $w(l, i, j)$ depends only on the label of peg $[l, i]$ and the label of peg $[l+1, j]$. We describe this weight as a function of labels shortly.

Each binary string is modified by adding the symbol $b$ between every two consecutive bits. A string of original length $n$ is then transformed into two consecutive (identical) levels of $2n - 1$
3.1. Pegs and Rubber Bands (Base Pair Model)

Figure 3.5: Reducing the instance of $k$-LCS, with $k = 3$, $S_1 = '0010111'$, $S_2 = '01010'$, $S_3 = '100101'$ to Pegs and Rubber Bands.

pegs each, where each peg is labeled by the corresponding symbol in \{0, 1, b\}. For any given integer $L$, the first and last levels consist of $L$ pegs labeled $\ast$. Therefore, we converted an LCS instance with $k$ strings in to a Pegs and Rubber Bands instance with $2k + 2$ levels. See Figure 3.5 for an illustration. We now define the weight as a function of labels:

- $w(0, 0) = w(1, 1) = w(b, b) = w(\ast, 0) = w(\ast, 1) = w(0, \ast) = w(1, \ast) = 1$
- $w(x, y) = -\infty$ otherwise

We now show that the strings have a common subsequence of length $L$ if and only if the optimal solution has a weight of $L + \sum_{i=1}^{k}(2n_i - 1)$, where $n_i$ is the original length of string $i$ and $k$ is the number of strings. Suppose the strings have a common sequence $R$ of length $L$. Wrap $L$ rubber bands around $\ast$’s in level 1 and symbols of $R$ in level 2, and then wrap rubber bands around symbols of $R$ in level 3 and level 4, and so on, until the last level, which has $L$ $\ast$’s, is reached. Ensure that the same pegs (in terms of positions) are used in in levels $i$ and $i + 1$ for $i \in \{2, 4, \ldots, 2k\}$. This contributes a weight of $(k + 1)L$. Then wrap rubber bands around unused pegs in level $i$ and $i + 1$ (which are mirror images), for $i \in \{2, 4, \ldots, 2k\}$. This adds a weight of $\sum_{i=1}^{k}(2n_i - 1 - L)$, making the total weight $L + \sum_{i=1}^{k}(2n_i - 1)$. This is optimal since
all pegs are being used. The converse follows from the fact that a weight of $L + \sum_{i=1}^{k}(2n_i - 1)$ is the maximum possible that can be achieved on a such an instance, and implies that all pegs are covered with rubber bands.

\[\square\]

### 3.2 Application to Multiple RNA Interaction

Figure 3.6: Multiple RNA interaction within the eukaryotic spliceosome, a large ribonucleoprotein assembly responsible for the excision of intervening sequences in precursor messenger (pre-mRNA) molecules. Shown is the spliceosomal U2-U6 small nuclear (snRNA) and introns I1 and I2 [SM95, ZBP+13].

Figure 3.6 shows how a complex of 4 RNAs can be represented as an instance of Pegs and Rubber Bands. With this formulation, we avoid the “locking” problem, since treating the RNAs pairwise would have favored the full binding of U2-U6 to include their left extremities (as in Figure 6.1), leaving I1 and I2 completely detached.

It should be noted here that although the above description maps one level to an entire RNA, we can place multiple RNAs on the same level by considering them as one long, concatenated RNA. If level $l$ contains multiple RNAs, the weight function $w(l, i, j)$ can be modified. For example, if level $l$ contains two RNAs $P$ and $Q$ of lengths $p$ and $q$ respectively, the quantities $w(l, i_1, j), \ldots, w(l, i_p, j)$ correspond to interactions between RNA $P$ and level $l + 1$, while the quantities $w(l, i_{p+1}, j), \ldots, w(l, i_{p+q}, j)$ correspond to interactions between RNA $Q$ and level $l + 1$. The order in which we place RNAs $P$ and $Q$ can affect the results.
3.3 Pegs and Rubber Bands (Loop Energy Model)

Note that the model allows interaction between different (but consecutive) levels. Arranging multiple RNAs on the same level precludes any interaction between them; however, it allows for more interaction across levels. In the rest of the chapter, we will assume without loss of generalization that each level has exactly one RNA. Therefore, we may use “RNA \( l \)” and “level \( l \)” interchangeably.

As is evident from the description of the framework, we require a given ordering of RNAs to stack them. This limits each RNA to interact with the RNAs immediately above and below it (again, wrapping around for the first and last RNAs). However, if an ordering is not provided, it is still possible to use a heuristic to try various orderings and attempt to find the ordering which maximizes the weight of the corresponding structure.

Recall that RNAs have directionality, and they interact in opposite directions. In Pegs and Rubber Bands, we have to alternate the directionality of each level. If we assume that an RNA on level \( l \) has the direction \( 5' \rightarrow 3' \), then RNAs on levels \( l - 1 \) and \( l + 1 \) must be placed in the \( 3' \rightarrow 5' \) direction. If we have an odd number of levels, then the first and last will have the same directionality, and therefore have no interactions between them.

3.3 Pegs and Rubber Bands (Loop Energy Model)

In the previous section, we described an application of Pegs and Rubber Bands to predicting multiple RNA interaction. The Pegs and Rubber Bands model described up to this point considers rubber bands to be independent of each other. When translating to multiple RNA interaction, each rubber band corresponds to an individual, independent basepair. In practice, however, basepairs are not independent, as we have already seen in Section 2.2, and structures should be scored using the Loop Energy model. For example, given the two short (sub)sequences \( 5' \ CAA \ 3' \) and \( 3' \ GUG \ 5' \), and unit weights between (C,G) and (A,U), both of the following are optimal solutions for Pegs and Rubber Bands:

\[
\begin{align*}
5' & \ldots C \ A \ A \ldots 3' & 5' & \ldots C \ A \ A \ldots 3' \\
| & \ | & | & \ \\
3' & \ldots G \ U \ G \ldots 5' & 3' & \ldots G \ U \ G \ldots 5'
\end{align*}
\]
3.3. Pegs and Rubber Bands (Loop Energy Model)  

The left window is $w(l, i_2, j_2, u, v)$ where $u = i_2 - i_1 + 1 = 3$ and $v = j_2 - j + 1 + 1 = 2$.

However, it is clear that the structure on the left has lower free energy, due to the stabilizing effect of stacked base pairs, whereas the structure on the right has a bulge, which is not as stable. In light of this, a more robust approach can incorporate the Loop Energy model, as well as an accompanying weight function. We achieve this by extending the Pegs and Rubber Bands framework to wrap a rubber band around a stretch of contiguous pegs in consecutive levels.

**Definition 3.5** (Windows and Intervals). A window $w(l, i_2, j_2, u, v)$ is a rubber band wrapped around pegs $(l, i_1), \ldots, (l, i_2)$, and $(l+1, j_1), \ldots, (l+1, j_2)$, where $i_2 \geq i_1$ and $j_2 \geq j_1$ are the last two pegs covered by the rubber band in level $l$ and level $l+1$, and $u = i_2 - i_1 + 1$ and $v = j_2 - j_1 + 1$ represent the length of the two intervals $[i - u + 1, i]$ and $[j - v + 1]$ in level $l$ and level $l + 1$, respectively.

In terms of RNA interaction, this represents interaction between bases in the range (or interval) $[i_1, i_2]$ on level $l$ and $[j_1, j_2]$ on level $l + 1$ (or 1 if $m = 1$). The weight contribution of placing such a rubber band is now given by $w(l, i_2, j_2, u, v)$. Figure 3.7 shows an example containing two windows.

The definitions of mixed graphs and overlaps require some modification in the context of windows; however, it is analogous to the definition given earlier. The mixed graph for a set of windows $S$ is created as follows:

- Intervals corresponding to windows of $S$ become the set of vertices,
- Windows of $S$ become undirected edges,
- Each ordered pair of intervals of $S$ on the same level becomes a directed edge.
We now say that $S$ has an overlap if the mixed graph of $S$ has a cycle or two intervals intersect (this replaces the earlier condition that a peg should not be covered by multiple rubber bands).

3.3.1 Weight Function

Some earlier interaction prediction algorithms aimed to minimize only the energy contribution by intermolecular basepairs or loops, depending on the model used. As discussed in Chapter 2, research in the area of (pairwise) RNA interaction has advanced beyond that point. Some of the newer directions take an approach based on partition functions to determine the likelihood that two regions on separate RNAs will interact. Similar to the method discussed in Chapter 2, we define the weight function for a window $w(l, i, j, u, v)$ as:

$$w(l, i, j, u, v) = RT \log P_l(\text{free}[i-u+1,i]) + RT \log P_{l+1}(\text{free}[j-v+1,j]) + RT \log Z^l_I(i-u+1,i,j-v+1,j)$$

where $P_l(\text{free}[i,j])$ is the probability that subsequence $[i,j]$ is free (does not fold) in RNA $l$, $Z^l_I(i_1,i_2,j_1,j_2)$ is the partition function of the interaction of subsequences $[i_1,i_2]$ in RNA $l$ and $[j_1,j_2]$ in RNA $l+1$ (subject to no folding within the RNAs subsequences), and $R$ and $T$ are the universal gas constant and temperature, respectively, as introduced in Section 2.3. In this sense, the weight of a window is the negative of its energy.

3.3.2 An Algorithm for Windows

The algorithm for the windows-based model is an extension of Algorithm 3.1. In fact, the first $m$ cases are the same, whereas the remaining cases consider non-overlapping windows in each level instead of pairs of pegs (i.e., rubber bands around single pegs in each level). Algorithm 3.2 shows the modified recurrence relation. As a heuristic, we also allow for the possibility of imposing a gap $g \geq 0$ between windows to establish a distance at which windows may be considered energetically separate. Algorithm 3.1 can be retrieved from Algorithm 3.2 if we set $u = v = 1$ and $g = 0$. 


Algorithm 3.2: Optimization Algorithm for PRB with windows

**Inputs**: $m$ levels

**Output**: Optimal weight

\[
T(i_1, i_2, \ldots, i_m) = \begin{cases} 
T(i_1 - 1, i_2, \ldots, i_m) \\
T(i_1, i_2 - 1, i_3, \ldots, i_m) \\
\vdots \\
T(i_1, \ldots, i_{m-1}, i_m - 1) \\
\max \\
T((i_1 - u - g)^+, (i_2 - v - g)^+, i_3, \ldots, i_m) + w(1, i_1, i_2) \\
T((i_1, i_2 - u - g)^+, (i_3 - v - g)^+, i_4, \ldots, i_m) + w(2, i_2, i_3) \\
\vdots \\
T((i_1, \ldots, (i_{m-1} - u - g)^+, (i_m - v - g)^+) + w(m - 1, i_{m-1}, i_m) \\
T((i_1 - u - g)^+, i_2, \ldots, i_{m-1}, (i_m - u - g)^+) + w(m, i_m, i_1) \ (\text{wrap around})
\end{cases}
\]

where $w(l, i, j, u, v) \in \mathbb{R}$, $T(0, 0, \ldots, 0) = 0$, and $g$ is the predefined gap size.

3.4 Approximation Algorithms

While the problem is NP-hard, we can show that it can be approximated as closely as desired (although with higher running time as the approximation becomes finer). In fact, we show that there is a Polynomial Time Approximation Scheme (PTAS) for Pegs and Rubber Bands.

**Definition 3.6** (Definition 1.2 in [WS11]). A polynomial-time approximation scheme (PTAS) is a family of algorithms \( \{A_{\epsilon}\} \), where there is an algorithm for each \( \epsilon > 0 \), such that \( A_{\epsilon} \) is a \((1+\epsilon)\)-approximation algorithm (for minimization problems) or a \((1-\epsilon)\)-approximation algorithm (for maximization problems).

Algorithm 3.3 computes a solution that has weight at least \( \frac{m - \lceil m/k \rceil}{m} \) times the optimal, where \( k < m \) is a parameter. We can show that Algorithm 3.3 actually represents a class of algorithms, which gives us a PTAS for Pegs and Rubber Bands by setting \( \epsilon \) as a function of \( k \).

**Theorem 3.2.** For every \( \epsilon > 0 \), there is an algorithm for PRB that yields a \((1-\epsilon)\)-approximation and runs in \( O(en^2 + \frac{1}{\epsilon} mn^O(1)) \) time.
Algorithm 3.3: A \( \frac{m - \lfloor m/k \rfloor}{m} \)-approximation algorithm for PRB

Inputs: \( m \) levels, an integer \( k < m \).

Output: APPROX: A solution with weight at least \( \frac{m - \lfloor m/k \rfloor}{m} \) times the optimal.

\begin{algorithmic}
\State \( a \leftarrow \lfloor m/k \rfloor \)
\State \( b \leftarrow m \mod k \)
\For \( i \in [0, m) \)
  \State // \( i \) represents a starting point of a new circular solution
  \State \( W_i \leftarrow 0 \)
  \For \( j \in [0, a) \)
    \State \( f \leftarrow i + jk \)
    \State \( \sigma_{ij} \leftarrow \) instance of PRB for \( k \) levels
    \State \( f \mod m, (f + 1) \mod m, \ldots, (f + k - 1) \mod m \) only
    \State \( W_j \leftarrow \) optimal solution of instance \( \sigma_{ij} \) using Algorithm 3.1 or Algorithm 3.2
    \State \( W_i \leftarrow W_i + W_j \)
  \EndFor
  \If \( b \neq 0 \)
    \State \( f \leftarrow i + \alpha k \)
    \State \( \sigma \leftarrow \) instance of PRB for \( \beta \) levels
    \State \( f \mod m, (f + 1) \mod m, \ldots, (f + \beta - 1) \mod m \) only
    \State \( W \leftarrow \) optimal solution of instance \( \sigma \) using Algorithm 3.1 or Algorithm 3.2
    \State \( W_i \leftarrow W_i + W \)
  \EndIf
\EndFor
\State APPROX \( \leftarrow \max_i W_i \)
\end{algorithmic}

Proof. For any given \( \epsilon > 0 \), set \( k \) to be an integer that is a function of \( \epsilon \). Algorithm 3.3 computes \( m \) different solutions, where each solution consists of several subproblems with at most \( k \) levels each. Each subproblem is solved independently, and their weights are summed to obtain a weight for each solution. The solution with the maximum weight is returned.

For the purpose of this proof, we relabel the levels 0 to \( m - 1 \) instead of 1 to \( m \). Let \( W[i, \ldots, j] \) denote the weight of the optimal solution of the subproblem consisting only of levels \( i, i+1, \ldots, j \mod m \). Note that \( i \) and \( j \) are not consecutive if \( j - i \neq 1 \mod m \), and that \( W[i \ldots i] = 0 \). Suppose \( m = ka + b \), where \( 0 \leq b < k \); i.e., \( a \) and \( b \) are the quotient and remainder, respectively,
when \( m \) is divided by \( k \). We now devise different solutions that consist of subproblems of size at most \( k \): there will be \( \lceil m/k \rceil \) such subproblems in each solution (\( a \) of them have \( k \) levels and one has \( b \) levels). Each different solution is created by choosing a new “starting point”. The first solution starts at level 0, the next at level 1, and so on.

Formally, define a solution \( W_i \) that consists of subproblems of size at most \( k \) as

\[
W_i = W[i \ldots i + k - 1] + W[i + k \ldots i + 2k - 1] + \ldots + W[i + (a - 1)k \ldots i + ak - 1] + W[i + m - b \ldots i + m - 1]
\]

The last subproblem in each solution is of size \( b \). If \( b = 0 \), set \( W[i + m - b \ldots i + m - 1] = 0 \).

E.g.,

\[
\begin{align*}
W_0 &= W[0 \ldots k - 1] + W[k \ldots 2k - 1] + \ldots + W[m - b \ldots m - 1] \\
W_1 &= W[1 \ldots k] + W[k + 1 \ldots 2k] + \ldots + W[m - b + 1 \ldots m] \\
&\vdots \\
W_{m-1} &= W[m - 1 \ldots m + k - 2] + W[m + k - 1 \ldots m + 2k - 2] + \ldots + W[2m - b - 1 \ldots 2m - 2]
\end{align*}
\]

In addition, define \( \text{OPT}[i] \) as the weight of the rubber bands between levels \( i \) and \( i + 1 \) mod \( m \) in the optimal solution of all \( m \) levels, and let \( \text{OPT} \) be the weight of that solution. Since \( W[i \ldots j] \) is a local optimal,

\[
W[i \ldots j] \geq \text{OPT}[i] + \text{OPT}[i + 1] + \cdots + \text{OPT}[j - 1]
\]

Next, we compute \( m \) solutions \( W_0, W_1, \ldots, W_{m-1} \). Each solution \( W_i \) is missing exactly \( \lceil m/k \rceil \) pairs of levels, since it consists of \( \lceil m/k \rceil \) subproblems, and is therefore “cut” at as many points. For example, \( W_i \) is missing the pairs \([k, k + 1], [2k, 2k + 1], \ldots\). In all, we have \( m \) solutions due
to circular shifts, so each pair is cut a total of \([m/k]\) times. Alternatively, we can say that each pair appears \(m - [m/k]\) times in these \(m\) solutions.

The idea is to decompose each solution into contributions by consecutive pairs of levels, and compare those to corresponding (pairwise) contributions in the optimal structure. Summing the weights of all \(m\) solutions and plugging in 3.2, we have

\[
\sum_{i=0}^{m-1} W_i \geq \sum_{i=0}^{m-1} (m - [m/k])OPT[i] = (m - [m/k])OPT
\]

Using the fact that \(m \max_i W_i \geq \sum_{i=0}^{m-1} W_i\), we have that

\[
\max_i W_i \geq \left(1 - \frac{[m/k]}{m}\right)OPT
\]

We can achieve the desired \((1 - \epsilon)\) factor approximation by making \(\frac{[m/k]}{m} \leq \epsilon\), which when \(m\) is large enough, can be done if \(k = O\left(\frac{1}{\epsilon}\right)\).

There are \(O(m)\) subproblems of at most \(k\) levels each. A subproblem requires a time polynomial in \(n = \max n_l\) for a fixed \(k\), \(O(kn^k)\), using Algorithm 3.1, where \(W(n_0, n_1, \ldots, n_k)\) is the optimal weight. Therefore, these \(O(m)\) subproblems can be precomputed in a total of \(O(mkn^k)\) time. Now each of the \(m\) solutions has \([m/k]\) subproblems, so the running time required to find all \(W_i\) is \(O(m^2/k)\) using the precomputed solutions to subproblems.

The total time required for this Algorithm 3.3 to run is \(O(m^2 n^k)\). However, if the first and last levels are not allowed to interact, then Algorithm 3.4 finds a \(\frac{k-1}{k}\)-approximation in \(O(mkn^k)\) time instead.

In Algorithm 3.4, as a practical step, and instead of using the \(W_i\)'s for the comparison, we can fill in for each \(W_i\) some additional rubber bands (interactions) between (RNAs) level \(i\) and level \(i + 1\), between level \(i + k\) and level \(i + k + 1\), and so on, by identifying the pegs of these levels that are not part of the solution. This does not affect the theoretical guarantee but gives
Algorithm 3.4: A \( \frac{k-1}{k} \)-approximation algorithm for PRB without wrap-around

**Inputs**: \( m \) levels, an integer \( k < m \).

**Output**: APPROX: A solution with weight at least \( \frac{k-1}{k} \) times the optimal.

1. **foreach** \( i \in [0, k) \) **do**
2. \( \sigma \leftarrow \) instance of PRB for \( i \) levels \( 0, 1, \ldots, i - 1 \) only
3. \( W \leftarrow \) optimal solution of instance \( \sigma \) using Algorithm 3.1 or Algorithm 3.2
4. \( W_i \leftarrow W \)
5. **while** \( f \leq m - k \) **do**
6. \( \sigma \leftarrow \) instance of PRB for \( k \) levels \( f, f + 1, \ldots, f + k - 1 \) only
7. \( W \leftarrow \) optimal solution of instance \( \sigma \) using Algorithm 3.1 or Algorithm 3.2
8. \( W_i \leftarrow W_i + W \)
9. \( f \leftarrow f + k \)
10. **end**
11. **if** \( f < m \) **then**
12. \( \sigma \leftarrow \) instance of PRB for \( k \) levels \( f, f + 1, \ldots, m \) only
13. \( W \leftarrow \) optimal solution of instance \( \sigma \) using Algorithm 3.1 or Algorithm 3.2
14. \( W_i \leftarrow W_i + W \)
15. **end**
16. **end**
17. \( \text{APPROX} \leftarrow \max_i W_i \)

a larger weight to the solution. We call it gap filling. We do not use this strategy when the last level is allowed to interact with the first, since it may result in crossings in the mixed graph.

**Theorem 3.3.** PRB without wrap-around admits a PTAS that runs in \( O(m^{\frac{1}{\epsilon}}n^{\frac{1}{2}}) \) time and yields a \((1 - \epsilon)\)-approximation.

**Proof.** Instead of \( m \) circular shifts, we only consider the following solutions of size at most \( k = O\left(\frac{1}{\epsilon}\right)\):

\[
W_i = W[0 \ldots i - 1] + W[i \ldots k + i - 1] + W[k + i \ldots 2k + i - 1] + \ldots
\]


3.4. Approximation Algorithms

(a) An input to Algorithm 3.4, that achieves an approximation factor of exactly $\frac{k-1}{k}$.

(b) An input to Algorithm 3.4, that contributes zero weight (in the limit) during gap filling.

(c) An example where $k = 3$. After both subproblems have been solved independently, black rubber bands cannot be selected during gap filling.

Figure 3.8: Adversarial inputs for Algorithms 3.4 and 3.3 to show that they cannot be guaranteed to achieve a better approximation factor for all instances.

There are $k$ such solutions (since $1 \leq i < k$), and each pair $[j, j + 1]$ is missing from exactly one solution. We can use an argument similar to the one in the proof of Theorem 3.2 to state that

$$\sum_{i=0}^{m-1} W_i \geq (k-1)OPT$$

The rest of the proof follows as before, and $\max_i W_i \geq \left(\frac{k-1}{k}\right)OPT$. There are $k \times \lceil \frac{m}{k} \rceil = O(m)$ subproblems, and each can be solved in $O(kn^k)$ time.

We now show that the analyses of the algorithms described above are tight.

**Corollary 3.1.** There are infinitely many instances where Algorithms 3.3 and 3.4 produce solutions which have weight exactly $\frac{k-1}{k}OPT$.

**Proof.** The input shown in Figure 3.8a consists of $m$ levels with 2 pegs each, with rubber bands placed in an alternating fashion. We first consider Algorithm 3.4, in which wrap around is not permitted. For this input, $OPT = m - 1$. Assume $k|m - 1$ and let $t = \frac{m-1}{k}$. If the size of the first subproblem in Algorithm 3.4 is $i$, the Algorithm will produce a solution consisting of subproblems of the following sizes: $i, k, \ldots, k, k-i+1$. The number of subproblems of size $k$ in this solution is $t - 1$. By construction, a subproblem of size $x$ has an optimal weight of $x - 1$, and
thus the sum of the optimal weights of the subproblems in each solution is \( t(k - 1) = \frac{k-1}{k}\text{OPT} \).

The instance in Figure 3.8a can be generalized by duplicating rubber bands in a similar pattern.

Now consider Algorithm 3.3, in which wrap around is permitted. Assuming \( m \) is an even number, modify the instance in Figure 3.8a, to wrap a rubber band around peg 2 of level 1 and peg 2 of level \( m \). In this case, \( \text{OPT} = m \), and we assume \( k|m \). Thus the approximation factor \( \frac{m-[m/k]}{m} \) reduces to \( \frac{k-1}{k} \). Each of the \( m \) solutions produced by Algorithm 3.3 contains exactly \( \frac{m}{k} \) subproblems of optimal weight \( k-1 \). Then the total weight of each solution is \( \frac{k-1}{k}\text{OPT} \).

A more interesting situation is when gap filling is allowed as a heuristic. In this case, consider the instance shown in Figure 3.8b, where the thick black rubber bands have weight 1 and the thin gray rubber bands have weight \( \epsilon \), where \( \epsilon > 0 \) is much smaller than 1. Assuming no wrap around, \( \text{OPT} = m - 1 + 2\epsilon \). With Algorithm 3.4, each subproblem consumes pegs in a way that prevents selection of black rubber bands in the gap filling stage. Figure 3.8c shows an example with two subproblems of size 3; the black rubber band cannot be selected in gap filling (between levels 3 and 4) since its corresponding pegs are covered with gray rubber bands, although in some instances, some gray rubber bands can be selected. Using a similar argument as before, we can show that as \( \epsilon \to 0 \), the approximation factor tends to \( \frac{k-1}{k} \).

\[ \Box \]

3.5 Heuristic for Optimal Ordering

We now describe how to relax the ordering condition on the interaction pattern of the RNAs. We first identify each RNA as being even (sense) or odd (antisense), but this convention can obviously be switched. Given \( m \) RNAs and a permutation on the set \( \{1, \ldots, m\} \), we map the RNAs onto the levels of a Pegs and Rubber Bands problem as follows: We place the RNAs in the order in which they appear in the permutation on the same level as long as they have the same parity (they are either all even or all odd). We then increase the number of levels by one, and repeat. RNAs that end up on the same level are virtually considered as one RNA that is the concatenation of all. However, in the corresponding Pegs and Rubber Bands problem, we do not allow windows to span multiple RNAs, nor do we enforce a gap between two windows in different
3.5. Heuristic for Optimal Ordering

---RNA 1---RNA 3---

---RNA 4---

---RNA 7---RNA 5---

---RNA 8---RNA 2---RNA 6---

Figure 3.9: Placement of the permutation 1, 3, 4, 7, 5, 8, 2, 6 where the RNA number also indicates its parity. The interaction pattern is less restrictive than before; for instance, RNA 7 can interact with RNA 2, RNA 4, RNA 6, and RNA 8.

RNAs. For example, if we consider the following permutation of RNAs \{1, 3, 4, 7, 5, 8, 2, 6\}, where the RNA number also indicates its parity (for the sake of illustration), then we end up with the following placement: RNA 1 and RNA 3 in that order on the first level, followed by RNA 4 on the second level, followed by RNA 7 and RNA 5 in that order on the third level, followed by RNA 8, RNA 2, and RNA 6 in that order on the fourth level, resulting in four virtual RNAs on four levels of pegs as shown in Figure 3.9.

But what is the best placement as a Pegs and Rubber Bands problem for a given set of RNAs? Figure 3.9 shows a possible (greedy) heuristic that tackles this question by starting with a random permutation and then searching for the best one via neighboring permutations (and recall that the permutation uniquely determines the placement).

To generate neighboring permutations for this heuristic algorithm one could adapt a standard 2-opt method used in the Traveling Salesman Problem (or other techniques). For instance, given permutation \(\pi\), a neighboring permutation \(\pi'\) can be obtained by dividing \(\pi\) into three parts and making \(\pi'\) the concatenation of the first part, the reverse of the second part, and the third part. In other words, if \(\pi = (\alpha, \beta, \gamma)\), then \(\pi' = (\alpha, \beta^R, \gamma)\) is a neighbor of \(\pi\), where \(\beta^R\) is the reverse of \(\beta\).

Algorithm 3.5 does not necessarily produce the optimal permutation, or guarantee that the output permutation would within some factor of optimal. However, some other algorithms do. A \(\frac{1}{2}\)-approximation can be obtained through a trivial algorithm using maximum matching. Let each RNA be a vertex, and let the weight of each edge \((u, v)\) between any pair of RNAs be the weight of interaction between \(u\) and \(v\) only (i.e., two levels). A maximum matching will then choose
3.6. Experimental Results

In this section we will present some of our experimental results on some known multiple RNA complexes, using the formulations presented so far. We delay consideration of other instances till Chapter 4, where we will present a generalized formulation and algorithm that can predict a wider class of RNA structures.

3.6.1 Structure Prediction

The human snRNA complex U6-U2 is necessary for the splicing of a specific mRNA intron [ZBP+13]. Only the preserved regions of the intron are considered, which consist of two structurally autonomous parts, resulting in an instance with a total of four RNAs. The computation
3.6. Experimental Results

Figure 3.10: The structure of the human snRNA complex U6-U2, as predicted by Algorithm 3.4, using a gap size of 4.

is performed using Algorithm 3.4 (i.e., without wrap around), with \( k = 2, 3, 4 \) and gap filling (as described in Section 3.4), and setting a gap of \( g = 4 \) between windows. In all three cases, the solution with the largest weight consistently finds the structure shown in Figure 3.6. This structure reveals a pattern consistent with the experimentally observed complex presented in [SM95, ZBP +13], and cannot be easily predicted by considering the RNAs in pairs; for instance, AUAC in U6 will bind to UAUG in both U2 and I1, and it is not immediately obvious which one to break without a global view, e.g. that AUGAU in U2 binds with UACUA in I2 as well. This is a typical issue of using local information to produce a globally optimal solution.

The 4-way junction construct of a hairpin ribozyme complex, obtained from [PHH +01], is shown in Figure 3.11. This structure of four RNAs has a circular interaction Substrate-RzA-FW1-FW2-Substrate (i.e., the bipartite interaction graph is a cycle). Helix H1 is predicted as a larger window, adding a C-G pair and a U-A pair (thus a bulge in H1 on the RzA side). These additional pairs unwind in the actual structure possibly due high order interactions with other parts of the structure. We will investigate the possibility of obtaining the correct structure as a suboptimal solution in Chapter 7.

We also predict the structure of the interaction complex of CopA and CopT, which are completely complementary. The structure has been of special interest to both biologists (e.g. [KMW +00, KESJ +00] and computer scientists [SBS10], due to its atypical behavior during interaction. Figure 3.13a shows the observed CopA-CopT complex. The complex undergoes certain tertiary transformations that delete previously existing base pairs. The interaction process starts with the regions shown in lower case (Figure 3.13b), but after other base pairs are formed, the
3.6. Experimental Results

Figure 3.11: A circular 4-way junction construct of a hairpin ribozyme complex [PHH⁺01].

complex transforms into a shape that causes these regions’ base pairs to be undone. This feature is called a reversible kissing loop. In the final, observed structure, these complementary regions participate neither in folding nor in interaction.

Figure 3.13c shows the predicted structure of CopA-CopT, which contains base pairs in the regions of the reversible kissing loop, due to the optimization nature of the problem. In fact, an optimization algorithm will predict the entire window, since it is entirely complementary, unless additional data or constraints are used to enforce a split. In addition, we have predicted a third window towards the right side of the structure, since it has a positive weight; such artifacts are a result of how we define weights, and dependent on tools that produce them. However, using heuristics similar to those in [SBS10, CBS09], we can predict a structure which does not contain this window. For example, if we use only the best 1000 windows (in terms of $RT \log Z^I_i (i - u + 1, i, j - v + 1, j)$), the optimal structure is the one shown in Figure 3.13d.

3.6.2 Fishing for Pairs

Six RNAs of which three pairs are known to interact are used [CBS09]. We are interested in identifying the three pairs. For this purpose, it will suffice to set $k = 2$ and use Algorithm 3.4 without gap filling. Furthermore, we only consider solutions in which each RNA interacts with
3.6. Experimental Results

at most one other RNA. The solution with the largest weight identifies the three pairs correctly (Figure 3.12). In addition, the interacting sites in each pair are consistent with the predictions of existing RNA-RNA interaction algorithms, e.g. [SBS10].
3.6. Experimental Results

(a) Observed base pairs in the CopA-CopT complex. This figure shows both the intermolecular as well as intramolecular base pairs.

(b) The intermolecular base pairs from (a), reproduced in this format to facilitate comparison between the observed structure and the predicted structure. The subsequences in lowercase are complementary, yet no base pairs form between them to geometric constraints.

(c) The predicted structure, using all data outputted by RNAup. The middle window is contiguous, and the structure contains an artifact towards the right. A computationally optimal structure would contain the additional base pairs in the middle window.

(d) With filtering heuristics similar to other programs, we can eliminate the rightmost window.

Figure 3.13: The observed structure of CopA-CopT, compared against results from Algorithm 3.2.
Chapter 4

Bipartite Pegs and Rubber Bands

4.1 Introduction

We now take a higher-level view of our model in order to generalize it. Our formulation of Pegs and Rubber Bands has considered levels stacked on top of each other, optionally allowing the first and last levels to interact. Since interaction occurs between a pair of levels, we can define the interaction pattern as a graph. For a given set of levels, let the interaction graph $IG = (V, E)$ be a graph that represents allowed interactions (i.e., it is a part of the input), where the vertices $V$ correspond to levels, and an edge $(u, v) \in E$ if and only if levels $u$ and $v$ are allowed to interact. Thus the interaction graph of the previously defined Pegs and Rubber Bands model is either a path or a cycle (when the first and last levels are allowed to interact).

Consider the circular 4-way junction ribozyme complex discussed in the previous chapter (Figure 3.11). The resulting interactions are bipartite in nature: The substrate and FW1 are in one class, and FW2 and RzA in the other, and each RNA in one class interacts with both RNAs in the other, making the graph complete bipartite. In order to predict the correct structure, it would have been necessary for the interaction graph to be a cycle.

The Spliceosomal U6/U2/Introns formation from Figure 3.6 offers another interesting example. The predicted structure involves interactions between I1-U6, U6-U2 and U2-I2. Thus the
The unrestricted interaction graph the Spliceosomal U6/U2/Introns complex is a cycle. A star interaction graph, where 1 is permitted to interact with all other RNAs/levels.

Figure 4.1: Two examples of interaction graphs, a cycle and a star.

resulting interaction is a path, which follows from the fact that only paths and cycles were permitted\(^1\). However, we could allow the interaction graph to be a cycle, so that interactions between I1 and I2 were also permitted. The interaction graph of this formulation is depicted in Figure 4.1a. It turns out that despite allowing the interaction to be cyclic, the optimal configuration is only a path.

It is natural to ask if we can predict structures based on more general interaction graphs. We mentioned earlier that when predicting RNA interaction structures, we can lay out multiple RNAs on the same level as long as they have the same parity (for example, as in Figure 3.9). This arrangement allows more general interactions, but suffers from the drawback that optimal solutions for different permutations have to be computed. The interaction graph in Figure 4.1b is a simple star graph that permits RNA 1 to interact with all other RNAs. Using the formulation from Chapter 3, we can put RNA 1 and RNA 4 on their own individual levels, and then put RNA 2 and RNA 3 on the same level. However, suppose the optimal structure involves an interaction between interval \([a, b]\) of RNA 1 and some interval of RNA 3, as well as interval \([c, d]\) of RNA 1 and some interval of RNA 2 (as depicted in the figure below), where \(a < b < c < d\). The arrangement on the left would not be able to predict this, since the windows overlap. However, the arrangement on the right will correctly predict it. Therefore, one would have to try both arrangements before being able to predict the optimal solution. The situation becomes more complex as more RNAs get involved, since this may require placing multiple RNAs on more than one level.

\(^1\)Note that when allowed interaction patterns are paths and cycles, the resulting structure could also be a set of paths.
We seek a more elegant solution that does not involve reconfiguring the layout. Note that we are still operating under the condition that each RNA must be classified as either even or odd. This limits the interaction graph to a bipartite graph, since even (resp. odd) RNAs cannot have an edge with another even (resp. odd) RNA. The interaction graphs shown in Figure 4.1 are both bipartite, and since any interaction graph corresponding to the original Pegs and Rubber Bands formulation is either a path or an even length cycle, it is naturally bipartite.

Under the bipartite graph model, windows need to be redefined:

**Definition 4.1 (Windows and Intervals).** For an even level \(l_1\) and odd level \(l_2\), a window \(w(l_1,l_2,i_2,j_2,u,v)\) is a rubber band wrapped around pegs \(i_1,\ldots,i_2\) in level \(l_1\), and pegs \(j_1,\ldots,j_2\) in level \(l_2\), where \(i_2 \geq i_1\) and \(j_2 \geq j_1\) are the last two pegs covered by the rubber band in level \(l_1\) and level \(l_2\) respectively, and \(u = i_2 - i_1 + 1\) and \(v = j_2 - j_1 + 1\) represent the length of the two intervals corresponding to the window in level \(l_1\) and level \(l_2\) respectively. In addition, we call the interval \([i_1,i_2]\) on \(l_1\) the even interval, and the interval \([j_1,j_2]\) on \(l_2\) the odd interval. For convenience, we may say that window \(w(l_1,l_2,i_2,j_2,u,v)\) is between levels \(l_1\) and \(l_2\).

An algorithm to solve the general bipartite model follows readily from the earlier windows based algorithm for the stacked Pairs and Rubber Bands. It is presented in Algorithm 4.1. The key difference is that instead of maximizing over pairs of adjacent levels, we maximize over all possible (even, odd) pairs. The running time of this algorithm is now \(O((m + |E|)n^m)\), where \(|E|\) is the number of edges in the interaction graph, but the space complexity is \(O(n^m)\) as before. Despite being exponential in the number of RNAs (or levels, since now each RNA is a level), it is practical for up to four RNAs.
4.2 Heuristic for Optimal Partitioning

In Section 3.5 we presented a heuristic algorithm to determine the optimal ordering of levels when the interaction pattern is a path. The bipartite graph model does not require an order on the RNAs, but it does require partitioning them (the levels) into even (the set $E$) and odd (the set $O$).

If we are given a set of RNA sequences without any parity, we must infer it. We use the following approach: Given $m$ RNAs, we start with a random permutation over $\{0, 1, \ldots, m\}$. All indices to the left of 0 belong to $E$, and all indices to the right of 0 belong to $O$. We find the optimal solution for Pegs and Rubber Bands given the complete bipartite interaction graph, i.e. $G = K_{|E|, |O|}$ (using Algorithm 4.1). Afterwards, a random search generates some neighboring permutations, e.g. using a 2-opt strategy similar to the algorithm in Section 3.5, and the same is repeated. When a better solution is revealed, the permutation is updated. When no better solution is found, we stop. The obtained solution represents a locally optimal one, and possibly the global optimal.

4.3 Experimental Results

To evaluate our algorithm, we compare its results against known secondary structures in the literature. Since the algorithm predicts interaction, we limit our analysis to intermolecular base pairs only; the intramolecular base pairs can be predicted by folding the unpaired nucleotides.
(known as constrained folding). Essentially, we have a set of intermolecular base pairs in the predicted structure \( P \), and a set of intermolecular base pairs in the biologically correct structure \( Q \). To measure the correctness of our algorithm, we compute the precision, recall, and F1 score, which are defined as follows:

\[
\text{Precision} = \frac{|P \cap Q|}{|P|} \quad \text{i.e., the fraction of predicted base pairs that are correct}
\]

\[
\text{Recall} = \frac{|P \cap Q|}{|Q|} \quad \text{i.e., the fraction of correct base pairs that are predicted}
\]

\[
\text{F1 score} = \frac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \quad \text{i.e., the harmonic mean of precision and recall}
\]

The F1 score gives an overall measure of the relevance of the results, and an F1 score close to 1 indicates that the predicted structure is close to the correct structure. Precision and recall have also been referred to as positive predictive value and sensitivity, respectively, in the literature.

We summarize our results in Table 4.1, and discuss some of them below. CopA-CopT and the human spliceosomal complex have already been discussed in the previous chapter.

<table>
<thead>
<tr>
<th>Complex</th>
<th>Our Results</th>
<th>Other Published Results</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Precision</td>
<td>Recall</td>
</tr>
<tr>
<td>CopA-CopT</td>
<td>0.600</td>
<td>1.000</td>
</tr>
<tr>
<td>CopA-CopT (filtered input)</td>
<td>0.730</td>
<td>1.000</td>
</tr>
<tr>
<td>Four-way junction construct of a hairpin ribozyme</td>
<td>0.953</td>
<td>1.000</td>
</tr>
<tr>
<td>Reverse-joined hairpin ribozyme HP-RJ</td>
<td>0.905</td>
<td>1.000</td>
</tr>
<tr>
<td>Yeast Spliceosomal U6 and U2 RNAs (truncated) with introns</td>
<td>1.000</td>
<td>1.000</td>
</tr>
<tr>
<td>Human Spliceosomal U6 and U2 RNAs</td>
<td>1.000</td>
<td>0.895</td>
</tr>
<tr>
<td>Human Spliceosomal U6 and U2 RNAs with introns</td>
<td>0.818</td>
<td>0.783</td>
</tr>
<tr>
<td>Conventional hairpin ribozyme HP-WT</td>
<td>0.913</td>
<td>1.000</td>
</tr>
<tr>
<td>Hammerhead ribozyme with HTF substrate and facilitators</td>
<td>1.000</td>
<td>1.000</td>
</tr>
</tbody>
</table>

Table 4.1: The precision, recall and F1 score of known RNA interaction complexes. Details are discussed separately (including sources for published results).

\(^{2}\)Here we compare our results against those in [SBS10], and the latter appear to be better, as we over-count base pairs. However, the CopT sequence presented in their paper is missing one critical base. According to our simulations, if the same malformed sequence is used in prediction, it gives more accurate results, as it predicts fewer base pairs.
We include three more examples of ribozyme complexes, where one RNA interacts with the rest of the RNAs, thus forming a star bipartite interaction graph (Figure 4.2). For the hammerhead ribozyme complex [JS96], the predicted structure is the correct solution. For the HP-WT and HP-RJ ribozyme complexes [SWM00], results are similar to those of the 4-way junction construct, in that the predicted structure contains a bulge which does not exist in the observed structure.

![Figure 4.2: Three more examples of ribozyme complexes [SWM00, JS96].](image)

4.4 A Special Case: Star Interactions

Recall that Theorem 3.1 states that Pegs and Rubber Bands for paths and cycles is $\text{NP}$-Hard, and this implies hardness for the most general bipartite graph model as well. However, it is worth investigating the hardness of other types of interaction graphs. The previous section shows the case of three interaction patterns which form star graphs. We refer to the optimization problem...
4.4. A Special Case: Star Interactions

(a) The center level is shown on the right. The first $n$ pegs, shaded black, represent the elements of the universe. In addition to these, there are $m - k = 2$ pegs, shaded grey, which we will refer to as blocking pegs. The figure on the right shows a level corresponding to the set $S_1 = \{1, 2, 3, 5\}$. Note that the first peg on the level is also a blocking peg.

(b) The dotted lines show blocking rubber bands whereas the solid lines represent membership rubber bands.

Figure 4.3: Set Cover reduction to Star-PRB for $\mathcal{U} = \{1, 2, 3, 4, 5, 6\}$, $\mathcal{C} = \{\{1, 2, 3, 5\}, \{2, 4, 6\}, \{1, 3, 5\}, \{4, 6\}\}$ and $k = 2$.

over star interactions graphs as Star-PRB. In this section, we show that Star-PRB is NP-Hard as well, via a reduction from the decision version of the set cover problem.

**Definition 4.2.** In the decision version of the Set Cover problem, we are given a universe of elements $\mathcal{U} = \{e_1, ..., e_n\}$, a collection of subsets of those elements $\mathcal{C} = \{S_1, S_2, ..., S_m\}$ where each $S_j \subseteq \mathcal{U}$, and an integer $k \leq m$. The goal is to find a collection of subsets $\mathcal{C}' \subseteq \mathcal{C}$ that covers all of $\mathcal{U}$ such that every element of $\mathcal{U}$ is in some set of $\mathcal{C}'$ and $|\mathcal{C}'| \leq k$.

**Theorem 4.1.** Star-PRB is NP-hard.

**Proof.** Given an instance of Set-Cover ($\mathcal{U}, \mathcal{C}, k$), we will construct an instance of Star-PRB. The center level of the star contains $n + m - k$ pegs as follows: one peg for each $e_i \in \mathcal{U}$ where the pegs are ordered by the element indices (i.e., peg $j$ corresponds to element $e_j$), followed by $m - k$ pegs where $m = |\mathcal{C}|$ and $n = |\mathcal{U}|$. We refer to these $m - k$ pegs as blocking pegs, and they appear to the right of the element pegs. We then construct a new level $L_j$ for each subset $S_j \subseteq \mathcal{C}$ with $n + 1$ pegs, where the first peg is a blocking peg, and the subsequent pegs correspond to elements in $S_j$, again ordered by the element indices. See Figure 4.3a for an example of this construction.
4.4. A Special Case: Star Interactions

We will now add weighted rubber bands (windows of lengths 1). Let $B$ denote the center level of the star. Connect element pegs in $B$ to corresponding element pegs in levels $L_j, 1 \leq j \leq m$ with rubber bands of weight 1, i.e., $w(B, L_j, i, p_i, 1, 1) = 1$ for $1 \leq i \leq n$, where $p_i$ is the index of the peg corresponding to the element $i$ on $L_j$. We will call these rubber bands **membership rubber bands**, since they denote which elements in the universe belong to a particular subset.

Next, connect each blocking peg in $B$ to the blocking peg of every level with a rubber band of weight $W_B$, which we will determine later, i.e., $w(B, L_j, n + i, 1, 1, 1) = W_B$, for all $1 \leq j \leq m, 1 \leq i \leq m - k$. We will call these rubber bands **blocking rubber bands**.

The idea is to assign a large value to $W_B$, so that the optimal solution contains $m - k$ blocking rubber bands, which is the maximum possible number of blocking rubber bands that can be selected. Each blocking rubber band overlaps with all membership rubber bands in the same level, thus “blocking” them from being selected. In other words, each blocking rubber band represents one subset $S_j$ that is not in the cover. By blocking (i.e., not selecting) $m - k$ subsets, the algorithm leaves $k$ subsets to represent a possible cover.

We claim that there is a cover of size $k$ if and only if the weight of the optimal solution is $n + (m - k)W_B$. Suppose that the optimal solution has weight $n + (m - k)W_B$. Since $W_B$ is large enough, the optimal solution will favor a blocking rubber band over all membership rubber band in any level, with the only constraint being the availability of blocking pegs. Thus an optimal solution blocks $(m - k)$ levels, leaving a weight of $n$ to be distributed evenly amongst unit weighted rubber bands in the $k$ remaining levels. These rubber bands represent a cover, since rubber bands corresponding to each of the $n$ elements are selected, and the size of this cover is $k$. For the converse, note that if there is a cover of size $k$ consisting of subsets $S_{j_1}, \ldots, S_{j_k}$, then it is possible to select membership rubber bands from levels $L_{j_1}, \ldots, L_{j_k}$ only. From the remaining levels $L_{j_{k+1}}, \ldots, L_{j_m}$, we select $m - k$ blocking rubber bands, for a total weight of $n + (m - k)W_B$. This weight is optimal.

To assign a value to $W_B$, we note that it only has to be larger than the maximum number of pegs in any level. Therefore, let $W_B = 1 + \max_{S \in \mathcal{C}} |S|$, which is $O(n)$, thus ensuring that our reduction is polynomial time. □
Figure 4.3b shows how an instance of SET COVER with $\mathcal{U} = \{1, 2, 3, 4, 5, 6\}, \mathcal{C} = \{S_i\}$ and $k = 2$, where $S_1 = \{1, 2, 3, 5\}, S_2 = \{2, 4, 6\}, S_3 = \{1, 3, 5\}$ and $S_4 = \{4, 6\}$, is transformed into a PRB-STAR instance. Note that there are two separate covers of size 2: $\{\{1, 2, 3, 5\}, \{4, 6\}\}$ and $\{\{1, 3, 5\}, \{2, 4, 6\}\}$. 
Chapter 5

Dependent Weights Pegs and Rubber Bands

In this chapter, we introduce the concept of dependent windows, and show how these dependencies may influence the weight of a structure. Since the weight function of a structure will be modified to incorporate dependencies, we refer to this formulation as Dependent Weights Pegs and Rubber Bands. Our scoring scheme has been relatively simple up till now, in that each window contributes an individual weight to the total weight of the structure. With dependent windows, we allow for the possibility of superadditive windows, i.e., two windows can together contribute a weight larger than the sum of their weights. As we will see in this chapter, this is attributed to the intervals of a window, i.e. it is related to the folding within RNAs rather than their interaction. We will discuss three different models of dependence that can be used to score a structure, which approximate the actual dependence to various degrees.

Two of the three models lead to NP-hard optimization problems, even for two levels. However, our goal in this chapter is to describe the new scoring schemes that incorporate dependence so that a given structure may be easily scored. This is important for the sampling approach presented in Chapter 7. Therefore, our primary focus is not on the optimization algorithms.
5.1 Motivation for Dependent Weights

To motivate our approach in this chapter, we consider two windows with the following weights:

\[ w(l, i_1, j_1, u_1, v_1) = RT \log P_l(\text{free}[i_1 - u_1 + 1, i_1]) + RT \log P_{l+1}(\text{free}[j_1 - v_1 + 1, j_1]) + RT \log Z_{l_1,l_2}^l(i_1 - u_1 + 1, i_1, j_1 - v_1 + 1, j_1) \]

\[ w(l_2, i_2, j_2, u_2, v_2) = RT \log P_l(\text{free}[i_2 - u_2 + 1, i_2]) + RT \log P_{l+1}(\text{free}[j_2 - v_2 + 1, j_2]) + RT \log Z_{l_1,l_2}^l(i_2 - u_2 + 1, i_2, j_2 - v_2 + 1, j_2) \]

If the two windows can coexist, they will contribute a weight equal to the sum \( w(l_1, l_2, i_1, j_1, u_1, v_1) + w(l_1, l_2, i_2, j_2, u_2, v_2) \) to any solution that contains both of them. However, \( \text{free}[i_1 - u_1 + 1, i_1] \) and \( \text{free}[i_2 - u_2 + 1, i_2] \) may not be independent events. Since RNA interaction involves unfolding RNAs as a preliminary step, the two intervals \([i_1 - u_1 + 1, i_1]\) and \([j_1 - v_1 + 1, j_1]\) may have been a part of the same folded substructure, and the unfolding of that substructure would have likely resulted in both intervals becoming free (e.g., due to the breaking of a stem). For instance, Figure 5.1 illustrates a scenario where these two subsequences are originally part of the folding in RNA \( l_1 \). Therefore, if \([i_1 - u_1 + 1, i_1]\) is free in RNA \( l \), it is reasonable to think that \([i_2 - u_2 + 1, i_2]\) is likely to become free as well. In other words, if \( P_l(\text{free}[i_2 - u_2 + 1, i_2] | \text{free}[i_1 - u_1 + 1, i_1]) > P_l(\text{free}[i_2 - u_2 + 1, i_2]) \), it should replace the latter.
5.1. Motivation for Dependent Weights

5.1.1 Previous Work

The idea of dependent intervals has been explored in a few papers previously. Chitsaz et al [CBS09] and Salari et al [SBS10] have developed algorithms to compute the joint probability of two intervals being free on a given RNA sequence, based on Mückstein et al’s algorithm in RNAup [MTH+06]. The idea is to compute the probability that \([a, b]\) is free by using a partition function algorithm that forces the interval \([c, d]\) to be unpaired at the same time. This computes the conditional probability that \([a, b]\) is free given that \([c, d]\) is free. The original algorithm in [MTH+06] runs in \(O(n^3w)\) time, where \(n\) is the sequence length and \(w\) is the maximum interval size that is allowed. Since this has to be done for each interval \([c, d]\), it adds a multiplicity factor of \(nw\) to the complexity. Considering triple joint probabilities adds another factor of \(nw\), and so on. Thus considering an arbitrary number of dependent intervals is prohibitive in terms of time complexity.

The algorithm in [CBS09] computes an approximation to the optimal interaction structure with probabilistic graphical models, whereas the algorithm in [SBS10] computes the optimal structure when an interval on one RNA is allowed to form base pairs with two (adjacent) intervals on the other RNA. Both algorithms use some sort of filtering on the input. In the case of CopA-CopT, discussed in Chapter 3, [SBS10]’s algorithm is not able to detect the split in the middle window. On the other hand, when we reproduced [CBS09]’s results, it appears that detecting the split requires a small upper bound on the window size (say 10), and it is not detected when the bound is higher.

When it comes to the prediction of the split of the reversible kissing loop of CopA-CopT, one has to use both dependent windows as well as sampling. Chapter 7 shows that this can be achieved using sampling and dependent weights, without any filtering.

5.1.2 Formulation

We now formulate the problem of Dependent Weights Pegs and Rubber Bands. If a solution contains two windows that define intervals \([a, b]\) and \([c, d]\) in level \(l\) with \(b < c\), then we may
5.1. Motivation for Dependent Weights

Figure 5.2: To consider all three dependencies, we must compute the joint probability of all three intervals being free, which adds another layer of complexity.

Consider them dependent in level $l$ (windows can be dependent in one or two levels), and thus add to their single weight contribution a new positive term for level $l$ given by $\Delta(l, a, b, c, d)$. One could then use

$$
\Delta(l, a, b, c, d) = RT \log P_l(\text{free}[a, b]) \\
+ RT \log P_l(\text{free}[c, d]|\text{free}[a, b]) \\
- RT \log P_l(\text{free}[a, b]) - RT \log P_l(\text{free}[c, d])
$$

(5.1)

as a possible alternative. For simplicity, we do not penalize the simultaneous inclusion of intervals $[a, b]$ and $[c, d]$ if $\Delta(l, a, b, c, d) \leq 0$. Therefore, we only consider cases where $\Delta(l, a, b, c, d) > 0$. In such cases, in addition to the single contribution of windows, a solution where both $[a, b]$ and $[c, d]$ of RNA $l$ interact will acquire more weight, due to the net positive effect of replacing the individual probabilities with the joint probability (through conditioning). When $a = b$ and $c = d$, we can refer to the dependency as an arc.

It is worth discussing the complexities that arise when one has multiple pairwise dependent intervals in a structure. Consider the three intervals in Fig. 5.2. To score the structure correctly, we may not simply add $\Delta(l, a, b, c, d) + \Delta(l, a, b, e, f) + \Delta(l, c, d, e, f)$, instead, we would need to compute

$$
P_l(\text{free}[a, b]) \times P_l(\text{free}[c, d]|\text{free}[a, b]) \times P_l(\text{free}[e, f]|\text{free}[c, d], \text{free}[a, b])
$$
5.1. Motivation for Dependent Weights

The last term adds another level of complexity since it requires conditioning on two intervals. The situation becomes more complicated as the number of intervals in a structure increases, as well as when more scenarios are considered.

Instead, we approximate the weight with an alternate strategy. When scoring a given structure, we only consider disjoint contributions, i.e., we allow an interval to participate in at most one dependency. In other words, we consider dependencies as a matching of intervals. Given a structure $S$, let $I_l(S)$ be the set of all intervals in level $l$ defined by windows in $S$, i.e., either $l_1 = l$ or $l_2 = l$ in $w(l_1, l_2, i, j, u, v) \in S$. Let $M_l(S)$ be the set of all matchings in $I_l(S)$. Then the modified weight of solution $S$ is defined as:

$$w(S) = \sum_{w(l_1, l_2, i, j, u, v) \in S} w(l_1, l_2, i, j, u, v) + \sum_{M \in M_l(S)} \max_{([a, b], [c, d]) \in M} \Delta(l, a, b, c, d) \tag{5.2}$$

We then seek a solution that maximizes the above. We will call this variant the \textsc{General-Dependent-Weights-PRB} problem. It is general because we do not specify any sort of structure on the dependencies. As we will see in the next section, it is \textsc{NP}-Hard to compute the optimal structure even if only one level has dependencies, with a total of two levels.

We include the general problem in our thesis out of theoretical interest. In Section 5.3, we will add a constraint on the structure of dependencies. This constraint is motivated by the way an RNA folds, so it is quite realistic. Unfortunately, with two levels, the problem remains \textsc{NP}-Hard to even approximate when dependencies are on both levels.

Finally, in Section 5.4, we introduce a very practical model for scoring dependencies. With a fixed number of levels, there is a polynomial time algorithm to compute the optimal structure. We will also describe a fast method to score a given structure.
5.2 General Dependencies

**Theorem 5.1.** \textsc{General-Dependent-Weights-PRB} is \textsc{NP-Hard} even when the number of levels is fixed at 2 and only one level has dependencies.

*Proof.* We make a reduction from 3-SAT. Say we are given a formula consisting of \( k \) clauses \( C_1, \ldots, C_k \) over \( n \) boolean variables as an instance of 3-SAT. We show how to transform it into an instance of \textsc{General-Dependent-Weights-PRB} with two levels, 0 and 1, using arcs. Recall that an arc is a dependency between two intervals of length 1. Our claim is that this instance of \textsc{General-Dependent-Weights-PRB} will have an optimal weight of \( nk + 2k \) if and only if the 3-SAT instance is satisfiable.

Every window in this construction will have unit weight, i.e., \( w(0,1,i,j,1,1) = 1 \). We first create variable gadgets. Each variable gadget consists of a set of \( 2k \) pegs \( (p_1, p_2, \ldots, p_{2k}) \) on each level. We refer to pegs \( p_1, \ldots, p_k \) on level 0 and pegs \( p_{k+1}, \ldots, p_{2k} \) on level 1 as the “true” pegs, and \( p_{k+1}, \ldots, p_{2k} \) on level 0 and pegs \( p_1, \ldots, p_k \) on level 1 as the “false” pegs. Next, we place windows between the \( i^{th} \) true peg in level 0 and the \( i^{th} \) true peg in level 1. That is, we create windows \( w(0,1,p_i,p_{k+i},1,1) \) for \( 1 \leq i \leq k \). We refer to these windows as “true” windows. We repeat the procedure for false pegs by creating windows \( w(0,1,p_{k+i},p_i,1,1) \) for \( 1 \leq i \leq k \). A variable gadget for \( k = 3 \) is shown in Fig. 5.3. The optimal solution will always place \( k \) rubber bands in each variable gadget, but either all will be true bands or all will be false. Therefore, if we have \( n \) variables (and thus \( n \) such gadgets), the weight contribution of all variable gadgets towards the weight of the optimum will be exactly \( nk \).

![Figure 5.3: A variable gadget (left) and a clause gadget (right).](image-url)
For a clause with \( q \) literals, create pegs \( P_1, \ldots, P_q \) on each level, and join peg \( P_i \) on level 0 with peg \( P_{q+1-i} \) on level 1, for \( 1 \leq i \leq q \), with a window of weight 1. Peg \( P_i \) on level 0 corresponds to the \( i^{th} \) literal in the clause. A clause gadget for 3 variables is shown in Fig. 5.3. Note that a clause contribute have a weight of exactly 1 in the optimal solution, regardless of the satisfiability of the formula.

We now connect clause gadgets with variable gadgets using unit weight arcs only on level 0, thus incorporating dependencies. The idea is to create an arc between each variable gadget and a clause gadget depending on whether the variable is positive or negative in that clause. Given \( C_j \), we do the following for each literal \( x_i \) in \( C_j \): if \( x_i \) is positive, add an arc between peg \( p_j \) of the variable of \( x_i \) and the corresponding peg of the clause gadget for \( C_j \); on the other hand if \( x_i \) is negative, add an arc between peg \( p_{k+j} \) of the variable of \( x_i \) and the corresponding peg of the clause gadget for \( C_j \).

We now prove our earlier claim. Note that regardless of whether the formula is satisfiable or not, the variable gadgets contribute \( nk \) and the clauses gadgets contribute \( k \) towards the optimal solution.

Suppose that the formula is satisfiable. Construct a solution by adding all true rubber bands of a variable if it is true in the satisfying assignment, or all false rubber bands otherwise. For each clause \( C_j \), we can choose any literal \( x \) that makes it true according to the satisfying assignment, and add the arc between \( x \)'s peg and \( C_j \)'s peg into our solution, as well the corresponding rubber band from \( C_j \). In this step, we added a weight of \( 2k \) for each clause. The total weight of the solution is \( nk + 2k \). The solution is optimal since there is maximum possible contribution by the variable bands, clause bands and the arcs.

For the converse, suppose that the optimal solution has score \( nk + 2k \). After subtracting variable and clause weights, we are left with \( k \), which has to come from arcs, since there are no more rubber bands that can be picked without crossings. Recall that an arc is included in a solution only if both of its corresponding rubber bands are included. This implies that a) all \( k \) arcs are from distinct clause gadgets, and b) there are \( k \) such non-conflicting variable rubber bands that connect to, and satisfy, a clause gadget.

\( \square \)
Despite the hardness of the optimization problem, maximizing the score for a given structure is actually quite trivial, since one only has to compute a matching over each set of intervals, as shown in Eq. 5.2.

5.3 Nested Dependencies

We now introduce a constraint on the dependencies in a structure: a matching of intervals must be nested. That is, if \([a, b], [c, d]\) and \([e, f], [g, h]\) belong to a matching (with \(b < c\) and \(f < g\)), then either \([a, d] \cap [e, h] = \emptyset\) or one is contained in the other. We call the version of our problem with this constraint the Nested-Dependent-Weights-PRB problem, and show that not only is it NP-Hard, but also that it has no constant factor approximation, unless \(P = NP\).

Lemma 5.1. Nested-Dependent-Weights-PRB is NP-Hard. Moreover, a constant factor approximation for it implies a constant factor approximation for the k-Longest Common Subsequence problem.

Proof. We make a reduction from the k-LCS problem, which was already introduced in Definition 3.4. Given \(n\) strings \(s_1, \ldots, s_n\), let \(L\) be the length of their longest common subsequence. We show how to construct an instance of Nested-Dependent-Weights-PRB with two levels, 0 and 1, that has an optimal weight \(OPT = L[nx - (n + 1)]\), where \(x\) is chosen such that

\[
\frac{n + 1}{n} < x < \frac{n}{n - 1}
\]

Furthermore, we show that any approximation to \(OPT\) consists of an integer multiple of \([nx - (n + 1)]\), say \(k\), and reveals a longest common subsequence of length \(k\).

We define \(s'_i\) to be string \(s_i\) reversed. If string \(s_i\) has length \(|s_i|\), then we call \(s_i[j]\) and \(s'_i[|s_i| - j + 1]\) duplicates (they represent two copies of the same character due to the reversal). We also define \(s_0 = s_1\) and \(s_{n+1} = s_n\). We then construct two levels of pegs, where each peg \(i\) in level \(l = 0, 1\) is represented by a character of some string. In the first level, we lay out the pegs given by the concatenated string \(s_1 s'_3 s_3 s'_5 \ldots\). In the second level, we lay out the pegs given
Figure 5.4: An example construction of a Nested-Dependent-Weights-PRB instance for \( s_1 = 0010111 \), \( s_2 = 01010 \), and \( s_3 = 100101 \), showing that \( \text{LCS}(s_1, s_2, s_3) = 4 \).

by the concatenated string \( s_0s'_1s_2s'_2s_3\ldots \). Figure 5.4 shows this construction for \( s_1 = 0010111 \), \( s_2 = 01010 \), and \( s_3 = 100101 \).

Every window in this instance has the form \( w(0, 1, i, j, 1) \) where peg \( i \) in level 0 and peg \( j \) in level 1 represent characters of \( s_{k+1} \) and \( s_k \) respectively for even \( k \) (\( s'_k \) and \( s'_{k+1} \) respectively for odd \( k \)) and the two characters are equal. We set \( w(0, 1, i, j, 1) = -1 \). In addition, we define \( \Delta(l, a, a, b, b) = x \) if pegs \( a \) and \( b \) of level \( l \) correspond to duplicate characters. If we represent windows as edges going across between level 0 and level 1, and dependence among windows (the \( \Delta \) terms) as arcs connecting duplicate characters in level 0 and in level 1, then every edge has weight \(-1\) and every arc has weight \( x \) (see Figure 5.4).

Since an arc can only contribute to a solution when its two corresponding edges are also included, we observe that the only way to achieve a positive weight is by a chain of length \( n + (n+1) \), consisting of an alternation of \( n \) arcs and \( n+1 \) edges, for a weight of \( nx - (n+1) > 0 \). In the example of Figure 5, the chain will consist of an edge from \( s_0 \) to \( s_1 \), an arc from \( s_1 \) to \( s'_1 \), an edge from \( s'_1 \) to \( s'_2 \), an arc from \( s'_2 \) to \( s_2 \), an edge from \( s_2 \) to \( s_3 \), an arc from \( s_3 \) to \( s'_3 \), and an edge from \( s'_3 \) to \( s'_4 \). By the choice of \( x \), any shorter chain will have negative weight. Furthermore, this chain represents one character common to all strings. The optimal solution will consist of \( \text{LCS} \) such chains that are nested as shown in Figure 5.4, and any approximation must contain \( k \) of them, for some integer \( k \leq L \). Therefore, any constant approximation to \( \text{OPT} \), say \( \alpha \), must score \( k[nx - (n+1)] \geq \alpha L[nx - (n+1)] \); resulting in \( k/L \geq \alpha \). This in turn means that we have a constant factor approximation for the Longest Common Subsequence problem. \( \square \)
The Longest Common Subsequence problem has been shown to be hard to approximate. A 1995 paper by Jiang and Li [JL95] proved that “There exists a constant $\delta > 0$ such that, if the LCS problem has a polynomial time-approximation algorithm with performance ratio $n^\delta$, where $n$ is the number of input sequences, then $P = NP$.” Their proof reduces the Max-Clique Problem to $k$-LCS, by creating a set of strings $S$ from the any input graph $G$ and showing that $G$ has a clique of size $k$ if and only if $S$ has a common subsequence of length $k$ for $k > 0$. The Max-Clique problem had been shown to have the above inapproximability result as a consequence of the celebrated 1992 result of Arora et al [ALM+92] on probabilistically checkable proofs. Jiang and Li thus showed that by extension, this limitation applies to LCS as well, due to the direct correspondence between Max-Clique and LCS.

However, in 2006-2007 it was shown by Zuckerman that “for all $\epsilon > 0$, approximating Max-Clique to within $n^{1-\epsilon}$ is NP-hard” [Zuc06], so this result is directly transferrable to $k$-LCS by Jiang and Li’s method. By Lemma 5.1, we have the following theorem.

**Theorem 5.2.** Nested-Dependent-Weights-PRB does not have a constant factor approximation unless $P = NP$.

While it is hard to find the optimal structure, we can still score a given structure using the Nested-Dependencies-PRB model. The algorithm is actually quite similar to Nussinov’s RNA folding algorithm. Suppose level $l$ of structure $S$ has the $n$ ordered intervals $I_1, I_2, \ldots, I_n$, where $I_k = [a_k, b_k]$. By definition, these interval are non-overlapping. We define the following recurrence to compute the optimal non-overlapping matching:

$$M(i, j) = \begin{cases} 
M(i + 1, j - 1) + \Delta(l, a_i, b_i, a_j, b_j) & \text{if } \Delta(l, a_i, b_i, a_j, b_j) \text{ is defined} \\
M(i, j - 1) + RT \log P_l(\text{free}[a_j, b_j]) \\
M(i + 1, j) + RT \log P_l(\text{free}[a_i, b_i]) \\
M(i, k) + M(k + 1, j) & \forall \ i < k < j - 1
\end{cases} \quad (5.3)$$

where $1 \leq i \leq j \leq n$ and $M(i, i - 1) = 0$, The optimal dependency score is given by $M(1, n)$. 
5.4 Adjacent Dependencies

To avoid the computation of conditional probabilities in $\Delta(l, a, b, c, d)$, we score structures using the adjacent dependencies model, in which we further restrict the dependencies in a structure by barring them from being nested. That is, if $[a, b]$ and $[c, d]$ are two dependent intervals in some structure, then there must be no third interval between these two. This allows us to adopt the following more practical definition.

$$
\Delta(l, a, b, c, d) = RT \ln P_l(\text{free}[a, d]) - RT \ln P_l(\text{free}[a, b]) - RT \ln P_l(\text{free}[c, d])
$$

(5.4)

when it’s positive. Thus we replace the individual free probabilities by one pertaining to the entire range. This will still capture dependence of two regions that separate from the stem of a hairpin loop of moderate size, as in the case of the window split in CopA-CopT. In general, when $\Delta(l, a, b, c, d)$ becomes sufficiently positive, it prevents such window splits from being detrimental to the total weight of the solution.

5.4.1 Scoring a Given Structure

The maximum no-overlap matching can now be computed easily using dynamic programming. Assume solution $S$ has $n$ intervals in level $l$ given by $[a_i, b_i]$, sorted from left to right for $i = 1, \ldots, n$. The weight of the maximum no-overlap matching in level $l$ can then be computed as $W_l(n)$ by dynamic programming, as shown below. The matching itself can be obtained by standard backtracking.

$$
W_l(i) = \max \left[ W_l(i - 1), W_l(i - 2) + \Delta(l, a_{i-1}, b_{i-1}, a_i, b_i) \right]
$$

(5.5)

where $W_l(0) = 0$, and $\Delta(l, a_{i-1}, b_{i-1}, a_i, b_i) = 1$ when not positive.
5.4.2 Computing the Optimal Structure

Unlike the general dependencies and nested dependencies models described previously, it is possible to compute the optimal structure with adjacent dependencies in polynomial time for two levels. A simple algorithm that compares each possible window to all adjacent intervals on both levels can be developed to compute the optimal structure. Assuming the lengths of both levels are $O(n)$, and the maximum window interval length is bounded by $w$, the complexity of this algorithm would be $O(n^4w^4)$. However, with additional bookkeeping, it is possible to devise an algorithm with complexity $O(n^3w^3)$. Using recursion diagrams [DP03, RE99, CSSB09], we first describe the algorithm assuming all intervals are unit length; we thus consider dependencies as arcs and windows as edges.

Figure 5.5: Decomposition of $W(i,j)$ into 6 types of subproblems.

Figure 5.5 shows possible decompositions of $W(i,j)$, which represents the optimal structure of the subproblem involving pegs $1, \ldots, i$ on level 0 and pegs $1, \ldots, j$ on level 1. The grey horizontal lines represent the levels, solid black lines represent edges, and dashed lines indicates that there may or may not be an edge. The pegs are colored according to the following scheme: black pegs may or may not have an edge or arc attached to them, red pegs have neither an edge nor an arc attached, blue pegs have an edge attached but no arc, and green pegs have both an edge and an arc attached. Cases (a), (b) and (c) in Figure 5.5 correspond to the original Pegs and Rubber
Bands formulation. (d) considers the case where \( i \) and \( j \) form an edge, and \( i \) also forms an arc with another peg to its left, where the latter subproblem is represented by \( U(i, j - 1) \). Case (e) and the \( V \) can be defined similarly. We postpone discussion of case (f) until later.

\[
U(k, l) \rightarrow P(k', l) + k' \quad \forall k' < k
\]

\[
V(k, l) \rightarrow Q(k', l') + \quad \forall l' < l
\]

Figure 5.6: Decomposition of \( U(k, l) \) and \( V(k, l) \).

Figure 5.6 describes the decomposition of \( U(k, l) \). We consider all possible pegs \( k' < k \), such that \( k' \) and \( k \) form an arc. This implies that \( k' \) must also form an edge; if \( k' \) forms an edge with \( k' \) on level 1, then we require that \( k' < k \). To keep the complexity of computing \( U(k, l) \) for fixed \( k, l \) linear, we delegate the task of computing the optimum structure with edge \( (k', l') \) to the subproblem represented by \( P(k', l) \). The matrix \( V(k, l) \) can be computed similarly.

\[
P(r, s) \rightarrow W(r - 1, s' - 1) + \quad \forall s' \leq s
\]

\[
W(r - 1, s' - 1) \rightarrow V(r - 1, s') + \quad \forall s' \leq s
\]

\[
Q(r, s) \rightarrow W(r' - 1, s - 1) + \quad \forall r' \leq r
\]

\[
W(r' - 1, s - 1) \rightarrow U(r', s - 1) + \quad \forall r' \leq r
\]

Figure 5.7: Decomposition of \( P(r, s) \) and \( Q(r, s) \).
The only restrictions on the structure in $P(r, s)$ are that 1) peg $r$ creates an edge with some peg $s' \leq s$, and 2) peg $r$ doesn’t form an arc. Peg $s'$, on the other hand, is free to form an arc with some peg $s''$, as long as $s'' < s'$. We therefore consider both possibilities. If $s'$ does form an arc, the subproblem is $V(r - 1, s')$, otherwise the subproblem is $W(r - 1, s' - 1)$. The matrix $Q(r, s)$ can be computed similarly.

![Diagram](image)

Figure 5.8: Expansion of case (f) of $W(i, j)$’s decomposition, as well as decomposition of $T(r, s)$.

We now revisit case (f) from the decomposition of $W(i, j)$, in Figure 5.8. Case (f) allows the possibility of both $i$ and $j$ forming arcs with pegs of the same edge. To keep the number of subcases linear, we iterate over pegs $j'$ in level 1 and delegate the task of computing the optimal structure with edge involving $j'$ to the subproblem $T(i, j')$. Note the $T(i, j)$ is different from $U(i, j)$ in that $T$ requires $i$ and $j$ to form and edge, whereas $U$ does not have any such requirement.

The recursion diagrams can now be expressed as recurrence relations, with arbitrary window sizes. We extend the definitions of $U, V, P, Q, T$ in the following way. Let

- $U(k, l, u)$ be the optimal solution to the subproblem bounded by $(k, l)$ where the interval $[k - u + 1, k]$ on level 0 forms an arc with an interval to its left.
- $V(k, l, v)$ be the optimal solution to the subproblem bounded by $(k, l)$ where the interval $[l - v + 1, l]$ on level 1 forms an arc with an interval to its left.

---

1 A subproblem bounded by $(k, l)$ is defined pegs $1, \ldots, k$ on level 0 and pegs $1, \ldots, l$ on level 1.
5.4. Adjacent Dependencies

- \( P(r, s, u) \) be the optimal solution to the subproblem bounded by \((r, s)\) where the interval \([r - u + 1, r]\) on level 0 forms a window with some pegs on level 1 in the range \([1, s]\).

- \( Q(r, s, v) \) be the optimal solution to the subproblem bounded by \((r, s)\) where the interval \([s - v + 1, s]\) on level 0 forms a window with some pegs on level 1 in the range \([1, r]\).

- \( T(r, s, u, v) \) be the optimal solution to the subproblem bounded by \((r, s)\) where the interval \([r - u + 1, r]\) on level 0 forms an arc with some interval \([r' - u' + 1, r']\), and the interval \([r' - u' + 1, r']\) forms a window with the interval \([s - v + 1, s]\) on level 1.

The recurrences in Algorithm 5.1 can be used to compute these terms. Note that for correctness, terms must evaluate to a positive real number or \(-\infty\). The optimal solution, then, is given by \(W(n_1, n_2)\), where \(n_1\) and \(n_2\) are the lengths of levels.
Algorithm 5.1: Optimization Algorithm for computing the optimal structure under the Adjacent Dependencies model.

**Output:** The optimal weight is given by \( W(n_1, n_2) \).

\[
\begin{align*}
U(k, l, u) &= \max_{k' < k \atop 1 \leq u' \leq w} P(k', l, u') + \Delta(0, k' - u' + 1, k', k - u + 1, k) \\
V(k, l, u) &= \max_{l' < l \atop 1 \leq u' \leq w} Q(k, l', v') + \Delta(1, l' - v' + 1, l', l - v + 1, l) \\
P(r, s, u) &= \max_{r' \leq r \atop 1 \leq u' \leq w} \begin{cases} \\
w(r, s', u, v) + W(r - u, s' - v) \\
w(r, s', u, v) + V(r - u, s', v)
\end{cases} \\
Q(r, s, v) &= \max_{r' \leq r \atop 1 \leq u' \leq w} \begin{cases} \\
w(0, 1, r', s, u, v) + W(r' - u, s' - v) \\
w(0, 1, r', s, u, v) + V(r', s - v, u)
\end{cases} \\
T(r, s, u, v) &= \max_{r' \leq r \atop 1 \leq u' \leq w} W(r' - u', s - v) + w(0, 1, r', s, u', v) + \Delta(0, r' - u' + 1, r', r - u + 1, r) \\
W(i, j) &= \max \begin{cases} \\
W(i - 1, j), W(i, j - 1), \\
\max_{1 \leq u \leq w} W(i, j, u - v) + w(0, 1, i, j, u, v), \\
\max_{1 \leq v \leq w} U(i, j, v - u) + w(0, 1, i, j, u, v), \\
\max_{1 \leq u \leq w} V(i, j, u - v) + w(0, 1, i, j, u, v), \\
\max_{1 \leq v \leq w} T(i, j', u, v') + w(0, 1, i, j, u, v) + \Delta(1, j' - v' + 1, j', j - v + 1, j)
\end{cases}
\end{align*}
\]
Chapter 6

Suboptimal Solutions I: Enumeration

We now discuss algorithms that predict multiple likely RNA complexes. We presented optimization algorithms in Chapters 3 and 4 that determine the “best structure”. From a computational perspective, other structures will be suboptimal. In this chapter, we will argue for the need to predict suboptimal structures of RNA interaction complexes, and provide one algorithm that produces these structures. In the next chapter, we will present another algorithm, based on sampling. The two algorithms are of different nature. The first is a generation algorithm that generates all possible structures with energy below a given threshold. The usefulness of this algorithm lies in counting structures with a particular shape. Here, our interest is in predicting the different shapes that an RNA complex may form, so for instance if two suboptimal structures differ by, say a single basepair, that difference is not considered substantial enough to declare two shapes. The second is a sampling algorithm, which draws structures from the Boltzmann distribution. This will give us a broad sense of the different possible structures that a set of RNA strands could form, without actually generating all of them. In both cases, due to the nature of the algorithms, the suboptimal solutions may be similar to each other, as mentioned above, so we define a distance function and use a clustering algorithm to filter the output into relatively distinct shapes.
6.1 Motivation and Background

There is an implicit assumption in predicting the structure with minimum free energy - that only a single structure exists and external conditions are unlikely to affect the structure formation process. However, several factors can influence the shape of the RBA complex. For example, the joint structure of spliceosomal RNAs U6 and U2 has been under considerable debate in the literature. Wet-lab experiments have yielded two different structures, which differ in the number of intermolecular helices. Figure 6.1 shows two proposed conformations of helix 1 in U6 and U2. It has been conjectured that the presence of protein may cause the intramolecular stem on U6 to form coaxial stacking [CC06], which leads to a longer stem, in turn causing the AGC triad to not form base pairs with the GCU triad in U2. Based on the literature, it is very likely that both structures could form.

It is possible for prediction algorithms to predict more basepairs than the actual observed structures. This is readily demonstrated by the case of the CopA-CopT complex. As discussed in Chapter 3, the structure undergoes certain tertiary transformations that delete previously existing base pairs, through the process of reversible kissing loop. Any secondary structure prediction algorithm that minimizes the free energy will add these reversible basepairs. Such an
Figure 6.2: Observed intermolecular base pairs in the CopA-CopT complex. The subsequences in lowercase are complementary, yet no base pairs form between them to geometric constraints. A computationally optimal structure would contain these base pairs.

algorithm does not typically incorporate geometric considerations. The observed CopA-CopT complex is shown again in Fig. 6.2.

Sampling and exhaustive generation have previously been employed as a means for computing suboptimal structures in the context of RNA folding. The challenge faced by sampling algorithms is that the space of RNA structures is too large to be sampled from directly. Ding and Lawrence [DL03, DCL04] developed a two step algorithm that first computes the partition function for all substrings of a given RNA sequence. To sample a single structure, it then recursively draws substructures from distributions conditioned on those substructures that have already been sampled. Metzler and Nebel [MN08], on the other hand, use a stochastic context-free grammar as a prior distribution for secondary structures. They incorporate certain types of pseudoknots in their model, and present a Markov Chain Monte Carlo method for sampling. Later, Waldispühl and Clote [WC07] extended the techniques of Ding and Lawrence to develop an algorithm to sample saturated structures, i.e., those structures which have a maximal number of base pairs, where adding further base pairs causes a crossing. To generate all suboptimal structures below a certain energy threshold, Wutchy et al [WFH+99] built on the earlier free energy minimization algorithms discussed in Section 2.4.

The algorithms we present here are fundamentally different in the sense that our problem involves multiple RNAs. The space of structures, already very large, grows exponentially with the number of levels. In our generation and sampling algorithms, we borrow some ideas from the literature, but apply them in novel ways to our situation.
In this chapter, we work with the generalized model of Bipartite Pegs and Rubber Bands introduced in Chapter 4. To keep the presentation simple, we use the simple additive weighting like that described in Chapters 3 and 4.

6.2 Generating Structures

Let $LB$, a real value in the range $[0, OPT]$, be the lower bound on the weight of the structures that we wish to generate. To efficiently generate all structures in the weight range $[LB, OPT]$, we employ a recursive strategy that does not create extraneous structures. Given a structure $S$ (whose weight could be less than $LB$), we want to be able to decide if $S$ is worth expanding. That is, we want to ask if more windows can be added to $S$ such that the total weight of the final solution will be at least $LB$.

Fortunately, we have discussed such an “oracle” already - the function $T(i_1, i_2, \ldots, i_m)$ from Chapter 4, Eq. 4.1. It computes the maximum weighted structure that can be created if we are limited to using pegs $1, \ldots, i_1$ on level 1, pegs $1, \ldots, i_2$ on level 2, and so on. In order to use this function, we first define the boundary of a structure $S$.

**Definition 6.1.** The boundary $B(S)$ of a structure (with $m$ levels) is an $m$-tuple $(i_1, i_2, \ldots, i_m)$, where $i_l$ is the smallest index at level $l$ such that peg $[l, i_l]$ in $S$ is covered by a window. If no peg in level $l$ is covered by a window, set $i_l = \infty$.

In other words, the boundary identifies the leftmost covered peg in each level. Let $(i_1, i_2, \ldots, i_m) = B(S)$, then the structure $S$ can be extended to a structure within the required weight range if $w(S) + T(i_1 - 1, i_2 - 1, \ldots, i_m - 1) \geq LB$, for example, by adding to $S$ all windows that correspond to $T(i_1 - 1, i_2 - 1, \ldots, i_m - 1)$. Of course, this does not mean that there is no other way to extend $S$ to increase the weight to the desired range, for example by inserting windows in some arbitrary order, so we cannot claim that this is an “if and only if” relationship. However, with a careful recursive strategy, we should be able to prune the tree at $S$ if $w(S) + T(i_1 - 1, i_2 - 1, \ldots, i_m - 1) < LB$. We will now develop such a strategy.
6.2. Generating Structures

Figure 6.3: Three structures with their boundaries and terminal windows shown in blue.

A Recursive Algorithm

In order for our recursive algorithm to be efficient, we must ensure that it does not create the same structure multiple times. Therefore, our goal is to impose some sort of order on the windows in a structure, i.e., there must be a unique way to generate a given set of windows. We can use the concept of the boundary to impose a notion of order: we only add a window to the ‘left’ of the boundary. Formally, given a structure $S$ with boundary $B(S) = (i_1, i_2, \ldots, i_m)$, a window $w(l_1, l_2, i, j, u, v)$ is to the ‘left’ of the boundary if $i < i_{l_1}$ and $j < i_{l_2}$. However, the condition that windows be added to the left of a structure is necessary but not sufficient to guarantee a unique way of generating structures. Consider the following example: we have two windows $w_1 = w(0, 1, i_0, i_1, u, v)$ and $w_2 = w(2, 3, j_2, j_3, u', v')$. The two windows don’t overlap, so the set $S = \{w_1, w_2\}$ is a valid structure. However, there are two ways of generating $S$ by adding a window to the left recursively: $\emptyset \rightarrow \{w_1\} \rightarrow \{w_1, w_2\}$ and $\emptyset \rightarrow \{w_2\} \rightarrow \{w_2, w_1\}$. Other simple ordering strategies, including ordering by levels, result in several ambiguities and drawbacks as well. Instead, we add a window $w$ to structure only if it is terminal in the resulting structure.

**Definition 6.2** (Terminal window). *Given a structure $S$ with boundary $B(S) = (i_1, i_2, \ldots, i_m)$, a window $w(l_1, l_2, i, j, u, v) \in S$ is terminal iff:

- $i - u + 1 = i_{l_1}$
- $j - v + 1 = i_{l_2}$
- no other window $w(l'_1, l'_2, i', j', u', v') \in S$ satisfies $i' - u' + 1 = i'_{l'_1}$, $j' - v' + 1 = i'_{l'_2}$ and $(l'_1, l'_2) > (l_1, l_2)$.
Algorithm 6.1: A recursive procedure to generate all structures with weight above a threshold.

**Inputs**: Threshold $\text{LB}$

**Output**: A collection of structures $\mathcal{C}$ with weight above the threshold

1. $\mathcal{C} \leftarrow \emptyset$
2. Procedure $\text{process}(S)$
   3. if $w(S) \geq \text{LB}$ then
   4. $\mathcal{C} \leftarrow \mathcal{C} \cup \{S\}$
   5. end
   6. foreach $w(l_1, l_2, i, j, u, v) \in \mathcal{W}$ that is terminal in $S \cup \{w\}$
      and $i < i_{l_1}$ and $j < i_{l_2}$ do
   8. $(i_1, i_2, \ldots, i_m) \leftarrow B(S \cup \{w\})$
   9. if $w(S \cup \{w\}) + T(i_1 - 1, i_2 - 1, \ldots, i_m - 1) \geq \text{LB}$ then
   10. $\text{process}(S \cup \{w\})$
   11. end
   12. end

where $>$ is the lexicographic order relation on ordered pairs.

Intuitively, a window is terminal in a structure $S$ if and only if it is the last window in $S$ that lies entirely on the boundary (i.e., intervals on both its levels constitute the boundary). The notion of ‘last’ is encoded in the definition by $(l_{1}', l_{2}') > (l_1, l_2)$, where we assume that pairs of levels have a fixed total order. Figure 6.3 shows some examples of terminal windows.

By definition, each structure has a unique terminal window. Therefore, our recursive strategy is to add a window $w$ to $S$ only if $w$ is terminal in $S \cup \{w\}$. If the resulting structure is not feasible (i.e., there is no way to extend it), the recursive step terminates. The entire procedure is listed in Algorithm 6.1.

Line 6 of Algorithm 6.1 deserves more attention, since a naive implementation of the `for` loop will test all windows each time. The first improvement we can make is to consider windows in only some pairs of levels, due to the following lemma.
Lemma 6.1. Suppose we have a structure $S$ with the terminal window $w^T = w(l^T_1, l^T_2, i^T, j^T, u^T, v^T)$. If another window $w = w(l_1, l_2, i, j, u, v)$ that lies to the left of the boundary $B(S)$ is terminal in $S \cup \{w\}$ then at least one of the following is true:

1. $l_1 = l^T_1$
2. $l_2 = l^T_2$
3. $(l_1, l_2) \geq (l^T_1, l^T_2)$

Proof. If $w$ is the terminal window in $S \cup \{w\}$, then it affected $w^T$ in one of two possible ways: either $w^T$ remains on the boundary of $S \cup \{w\}$, or it does not. In the former case, $w$ can be the terminal window only if $(l_1, l_2) \geq (l^T_1, l^T_2)$, which follows from the definition of terminal windows. In the latter case, the boundary changed in a way that it no longer touches both intervals of $w^T$, which could only happen if an interval belonging to $w$ was inserted on one of the levels of $w^T$, i.e., $l^T_1$ or $l^T_2$. Since the only levels being modified due to addition of $w$ are $l_1$ and $l_2$, this means that either $l_1 = l^T_1$ or $l_2 = l^T_2$ (or both).

The converse of the above is not always true in Bipartite Pegs and Rubber Bands. If a window $w(l_1, l_2, i, j, u, v)$ satisfies at least one of the first two conditions, it may not necessarily be the terminal window in $S \cup \{w\}$. If it satisfies the third condition, then it is terminal. Note that in the special cases when the bipartite graph is a path or cycle, a window is terminal if and only if it satisfies at least one of the three conditions.

As a first step, we partition windows by their levels in the preprocessing phase. Let $\mathcal{W}_{l_1, l_2} \subseteq \mathcal{W}$ be the set of all windows whose even interval is on $l_1$ and odd interval is on $l_2$. Now instead of looping over all windows in $\mathcal{W}$, we first identify the terminal window $w^T$ in $S$, then only loop over windows in those $\mathcal{W}_{l_1, l_2}$ where at least one of the three conditions of Lemma 6.1 are met. This is a heuristic to eliminate some candidates for terminal windows. Since the converse of Lemma 6.1 is not always true in the general case, a window considered via this method is not guaranteed to be terminal.
Secondly, we observe that for a particular pair of levels \((l_1, l_2)\), we do not have to test all windows in \(W_{l_1, l_2}\), due to a property of the matrix \(T\). For a window \(w(l_1, l_2, i_{l_1}, i_{l_2}, u, v)\),

\[
T(i_1, \ldots, i_m) \geq T(i_1, \ldots, i_{l_1} - u, \ldots, i_{l_2} - v, \ldots, i_m) + w(l_1, l_2, i_{l_1}, i_{l_2}, u, v)
\]  

(6.1)

We use this property in the following way. Suppose we have a structure \(S\) with boundary \(B(S)\), and two windows \(x = w(l_1, l_2, i_1, i_2, u, v)\) and \(y = w(l_1, l_2, i'_1, i'_2, u', v')\) with positive weight, such that \(i' \leq i - u\) and \(j' \leq j - v\) (i.e., \(y\) lies on the left of \(x\)). Then if \(S \cup \{x\}\) is not extendible, neither is \(S \cup \{y\}\), since \(w(S \cup \{x\}) + T(i_1 - 1, \ldots, i_{l_1} - u, \ldots, i_{l_2} - v, \ldots, i_m - 1)\) already covers the weight of window \(y\), by Eq. 6.1.

6.3 Distance function

One of our goals behind generating suboptimal structures is to determine functionally distinct structures. The functions of RNA complexes are governed by their shapes, which means we are interested in the different, distinct shapes that a given set of sequences can form. Many of the results of the generation algorithm have very similar shapes. Therefore, we consider dividing the output of our generation algorithm into a set of disjoint classes, and choose one representative structure from each class. We achieve this by developing a distance function, which we then use to cluster the output structures into their respective classes.

Fig. 6.4 shows three structures that are very similar to each other. Two of them are missing only three base pairs, but these missing base pairs do not cause the entire shape to be altered, since there are windows in exactly the same places, of more or less the same size. Therefore, the
three structures should belong to the same class, and we can represent the class using its most optimal structure. We call this structure the *representative* of the class.

We can use some standard clustering algorithms to cluster structures into classes. However, many of these algorithms, for instance k-means, assume the input to be a collection of points in Euclidean space. To this end, we would have to represent each RNA complex as a vector such that Euclidean distance between two vectors is meaningful, which can be challenging. On the other hand, a distance function that is not restricted to a particular space is desirable, since it allows us to use different clustering algorithms, such as hierarchical clustering. Since structures are sets of windows, we develop a distance function based on the Jaccard similarity index. The Jaccard similarity index for two sets $A$ and $B$ is defined as follows:

$$
J(A, B) = \frac{|A \cap B|}{|A \cup B|}
$$

and the corresponding Jaccard distance is defined as

$$D_J(A, B) = 1 - J(A, B)$$

By itself the Jaccard distance is not sufficient. Two different windows can be very similar if they have many nucleotides in common. However, $A \cap B = \emptyset$ unless all the windows in $A$ are exactly the same as those in $B$. Thus the distance function should be based on a method to compare windows as well. In addition, consider the two following structures:

```
..UUUCGUACUCGCCAAAGUUGAAG...
               |||||***********
..AAAGCAUGAGCGGUUUCAACUUUC...
               ||YYYYYYYYYYYYYY
```

If we base the distance on, say, the number of base pairs in a structure, then the distance here is small ($\sim 0.35$), even though these are two separate shapes. These limitations demand a more robust distance function.

To define our similarity index, we make use of the generation algorithm described in the previous section. If the two solutions are similar, we expect to have added a similar set of
6.3. Distance function

Given a structure $S$, let

$$w(l_1, l'_1, i_1, j_1, u_1, v_1), \ldots, w(l_{|S|}, l'_{|S|}, i_{|S|}, j_{|S|}, u_{|S|}, v_{|S|})$$

be the $|S|$ windows in the order that they were added to $S$ by Algorithm 6.1.

Each of these windows, say $w(l, l', i, j, u, v)$, defines the two intervals, $[i - u + 1, i]$ in level $l$ and $[j - v + 1, j]$ in level $l'$. Consider the set of interaction intervals $I(S) = \sum_l I_l(S)$ to be ordered accordingly. Therefore,

$$I(S) = \{I_1, \ldots, I_{2|S|}\} = \{[i_1 - u_1 + 1, i_1], [j_1 - v_1 + 1, j_1], \ldots, [i_{|S|} - u_{|S|} + 1, i_{|S|}], [j_{|S|} - v_{|S|} + 1, j_{|S|}]\}$$

is an ordered set of $2|S|$ intervals. Let $L(S) = \{(l_1, l'_1), \ldots, (l_{|S|}, l'_{|S|})\}$ be an ordered set of $|S|$ pairs, where $(l_i, l'_i)$ is the pair defining the $i^{th}$ window. Therefore, $L(S)$ means that we have the following set of pairwise interactions (not necessarily unique in terms of RNAs): RNA $l_1$ with RNA $l'_1$, RNA $l_2$ with RNA $l'_2$, \ldots, RNA $l_{|S|}$ with RNA $l'_{|S|}$. Two solutions that do not agree on this set are considered completely dissimilar; otherwise, their distance is given by the amount of overlap in their interaction intervals (as in the Jaccard metric [Jac01]), hence the following definition of distance:

Given two solutions $S_1$ with $I(S_1) = \{I_1, I_2, \ldots\}$ and $S_2$ with $I(S_2) = \{T_1, T_2, \ldots\}$, the distance between $S_1$ and $S_2$ is

$$d(S_1, S_2) = \begin{cases} 1 - \frac{\sum |I_i \cap T_i|}{\sum |I_i \cup T_i|} & L(S_1) = L(S_2) \\ 1 & \text{otherwise} \end{cases}$$

(6.3)

where intervals are treated as sets of integers.

**Theorem 6.1.** The distance defined above is a metric.

**Proof.** The distance is non-negative and symmetric. Furthermore, $d(S_1, S_2) = 0$ iff $S_1 = S_2$ because $|I_i \cap T_i| \leq |I_i \cup T_i|$, with equality iff $I_i = T_i$. Therefore, we only worry about proving
subadditivity (triangular inequality). Consider the two solutions $S_1$ and $S_2$, and a third solution $S_3$. If $L(S_1) \neq L(S_2)$, then $L(S_1) \neq L(S_3)$ or $L(S_2) \neq L(S_3)$. Without loss of generality, let $L(S_1) \neq L(S_3)$. This means $d(S_1, S_2) = 1$ and $d(S_1, S_3) = 1$ and thus $d(S_1, S_2) \leq d(S_1, S_3) + d(S_3, S_2)$. If $L(S_1) = L(S_2)$, then either $L(S_1) = L(S_2) \neq L(S_3)$ or $L(S_1) = L(S_2) = L(S_3)$. In the former case $d(S_1, S_3) = d(S_3, S_2)$, and thus $d(S_1, S_2) \leq d(S_1, S_3) + d(S_3, S_2)$. In the latter case, consider the sets

\begin{align*}
A &= \bigcup_i I_i \times \{i\} \\
B &= \bigcup_i T_i \times \{i\}
\end{align*}

where the intervals are treated as sets of integers. Observe that $d(S_1, S_2) = 1 - ((A \cap B)/(A \cup B))$, which is a Jaccard distance and is known to be a metric. \qed
Chapter 7

Suboptimal Solutions II: Sampling

7.1 Introduction

We use sampling as our second strategy to generate suboptimal structures. As discussed in Section 2.3, an RNA complex $S$ has a probability proportional to $\exp[-E(S)/RT]$ (eq. 2.1), and the partition function $Z = \sum_S \exp[-E(S)/RT]$ serves as the normalization constant. To determine suboptimal RNA complexes, we can draw samples from the Boltzmann distribution of these RNA complexes.

For any sampling algorithm for multiple RNA interaction, it is crucial that it samples structures without explicitly generating the entire state space, which is exponential in the number of windows between all RNAs. In light of this, we have developed an approach based on Gibbs sampling and the Metropolis-Hastings algorithm. With Gibbs sampling, we are able to incrementally explore the state space for every pair of levels. We then use Metropolis-Hastings to sample structures from a particular pair.

7.1.1 Gibbs Sampling

Gibbs sampling is a Markov Chain Monte Carlo algorithm to generate samples from joint distributions when direct sampling is prohibitive. First introduced by Geman and Geman in [GG84],
and named after the physicist Josiah Willard Gibbs, the Gibbs sampler works by sampling each random variable individually in order, conditioned on the current values of the other variables. Suppose we want to sample \( X = (x_1, x_2, \ldots, x_n) \) from the joint distribution \( P(x_1, x_2, \ldots, x_n) \). To generate the \( i \)th sample, say \( (x_1^{(i)}, x_2^{(i)}, \ldots, x_n^{(i)}) \), the Gibbs sampler draws \( x_1^{(i)} \sim P(X_1 = x_1 \mid X_2 = x_2^{(i-1)}, \ldots, X_n = x_n^{(i-1)}) \), then draws \( x_2^{(i)} \sim P(X_2 = x_2 \mid X_1 = x_1^{(i)}, X_3 = x_3^{(i-1)}, \ldots, X_n = x_n^{(i-1)}) \), and so on until it draws \( x_n^{(i)} \sim P(X_n = x_n \mid X_1 = x_1^{(i)}, X_2 = x_2^{(i)}, \ldots, X_{n-1} = x_{n-1}^{(i)}) \). Therefore, when sampling variable \( x_j \) in iteration \( i \), the prior consists of values of variables \( x_1, \ldots, x_{j-1} \) obtained in iteration \( i \) and values of variables \( x_{j+1}, \ldots, x_n \) in iteration \( i - 1 \). Under typical conditions of ergodicity \([DEKM98]\), the Gibbs guarantee is that \( (x_1^{(t)}, x_2^{(t)}, \ldots, x_n^{(t)}) \) for large \( t \) is a sample from \( P(x_1, x_2, \ldots, x_n) \).

We employ Gibbs sampling for multiple RNA interaction by adopting the following setting for the general bipartite model of Pegs and Rubber Bands. Suppose we are given the bipartite graph \( G = (E, O, E) \) where \( E \) is the set of even levels, \( O \) is the set of odd levels, and \( E \) is the set of permitted interactions. For some structure \( S \), we use the variable \( S_{l_1, l_2} \) to represent the set of windows between levels \( l_1 \) and \( l_2 \) in \( S \), where \( l_1 \in E \) and \( l_2 \in O \). \( S \) can now be rewritten as \( S = \bigcup_{(l,l') \in G} S_{l,l'} \). If \( (l_1, l_2) \not\in G \), then \( S_{l_1, l_2} = \emptyset \).

Algorithm 7.1 generates samples from the joint distribution \( P(\bigcup_{(l,l') \in G} S_{l,l'}) \) using Gibbs sampling. At each time step \( t \), we iterate over all pairs of even/odd levels \( (l_1, l_2) \), and draw a sample from the distribution conditioned over the current structure \( \bigcup_{(l,l') \in G-(l_1,l_2)} S_{l,l'} \). That is, we sample from \( P(S_{l_1,l_2} \mid \bigcup_{(l,l') \in G-(l_1,l_2)} S_{l,l'}) \). It now remains to be seen how to draw samples from this conditional distribution.

The set \( S' = \bigcup_{(l,l') \in G-(l_1,l_2)} S_{l,l'} \) is a valid structure, and the state space for \( \bigcup_{(l,l') \in G-(l_1,l_2)} S_{l,l'} \) consists of structures that have the same windows as \( S \) in all levels except \( (l_1, l_2) \). For \( (l_1, l_2) \) our choice of windows is limited to exactly those that do not cause any overlaps in \( S \). Therefore, we
Algorithm 7.1: Gibbs Sampling to Generate Pegs and Rubber Bands’ Structures

**Inputs**: Set of even and odd levels, bipartite graph $G$, and an integer $T$.

**Output**: A sequence of samples $S^{(1)}, S^{(2)}, \ldots, S^{(T)}$.

1. Initialize $S[(l_1, l_2)] \leftarrow \emptyset \ \forall (l_1, l_2) \in G$
2. for $t=1$ to $T$ do
3.   foreach $(l_1, l_2) \in E$ do
4.      $S_{l_1, l_2} \sim P(S_{l_1, l_2} | \cup_{(l', l'') \in G - (l_1, l_2)} S_{l', l''})$ using Metropolis-Hastings
5.   end
6.   $S^{(t)} \leftarrow \cup_{(l', l'') \in G} S_{l', l''}$
7. end

can assume that

$$P(S_{l_1, l_2} = X | \cup_{(l', l'') \in G - (l_1, l_2)} S_{l', l''}) \propto \begin{cases} 0 & \text{contains overlaps} \\ \cup_{(l', l'') \in G - (l_1, l_2)} S_{l', l''} \cup X & \text{otherwise} \\ e^{w(X)/RT} & \text{otherwise} \end{cases}$$

where the exponential term is consistent with the standard Boltzmann distribution for the interaction of RNAs $l_1$ and $l_2$, knowing that $w(X)$ represents the negative of the energy multiplied by $RT$. Recall from Chapter 5 that it is trivial to incorporate dependencies in a given structure, since one has to find the best matching of intervals (under one of the three models) for each level. Therefore, in this chapter, the weight function $w(X)$ includes the contributions of dependencies in structure $X$.

### 7.1.2 Metropolis-Hastings

We have described how to employ Gibbs sampling to divide the state space into smaller segments that can be sampled iteratively. The state space for a pair of levels $(l_1, l_2)$ depends only on windows of the form $w(l_1, l_2, i, j, u, v)$ which also do not create overlaps in the overall conditional structure. This set is still too large to sample from directly, so we need to develop a local exploration strategy. The Metropolis-Hastings algorithm is suitable to sample structures in this
fashion. It is a Markov Chain Monte Carlo method which was described in [22] and [14], and since then has been utilized extensively in the literature.

The Metropolis-Hastings algorithm works by generating a chain of samples \( \sigma^{(1)}, \sigma^{(2)}, \ldots, \sigma^{(T)} \), where the move from \( \sigma^{(t)} \) to \( \sigma^{(t+1)} \) is governed by a proposal distribution and an acceptance probability. At time step \( t + 1 \), a candidate state \( \sigma^C \) is drawn from the conditional distribution \( Q(\sigma^C|\sigma^{(t)}) \), and then accepted or rejected with a probability based on the Metropolis-Hastings criteria, defined as

\[
\alpha(\sigma^C|\sigma^{(t)}) = \min \left\{ 1, \frac{Q(\sigma^{(t)}|\sigma^C)}{Q(\sigma^C|\sigma^{(t)})} \times \frac{\pi(\sigma^C)}{\pi(\sigma^{(t)})} \right\}
\]

(7.2)

where \( \pi(\sigma^{(t)}) \) is the probability of state \( \sigma^{(t)} \) in the target distribution. If the candidate state is accepted, we set \( \sigma^{(t+1)} = \sigma^C \), otherwise we set \( \sigma^{(t+1)} = \sigma^{(t)} \).

What makes Metropolis-Hasting attractive for a situation such as ours is that the acceptance probability is a function of the ratio of two probabilities in the target distribution. This obviates the need to compute the partition function (i.e., the normalization constant), which would have to be recomputed each time we revisit a pair of levels. The recomputation would have been necessitated by the fact that the Gibbs sampler in Algorithm 7.1 changes the set of permitted windows after each round. However, the acceptance probability only requires a term proportional to the target probability, which in our case is \( \exp[w(S)/RT] \) for structure \( S \).

7.2 Proposal Distribution

Vis-a-vis Metropolis-Hastings, our state space consists of structures which differ from each other only by windows between levels \( l_1 \) and \( l_2 \). The set of windows is further limited by the fact that we must not have overlaps in the overall structure by the addition of a window. Suppose the set of permitted windows in this particular round (i.e., line 4 of Algorithm 7.1) is \( \mathcal{V} \). For the proposal distribution, we define a neighborhood of some structure \( S^{(t)}_{l_1,l_2} \) in the following way. Note that we may use “state” and “structure” interchangeably, since each state of the underlying Markov chain \( \{S^{(t)}_{l_1,l_2}\}_t \) is just a structure.
Figure 7.1: The structure on the left has two windows that do not have a gap between their intervals on either level. We consider such windows as overlapping. On the other hand, the two windows in the structure on the right are not considered overlapping since there is a gap on level $l_2$.

**Definition 7.1 (Neighboring Structures).** A state $T_{l_1,l_2}$ is a neighbor of a structure $S_{l_1,l_2}$ if it can be created by one of the following three operations

1. A window $w(l_1,l_2,i,j,u,v) \in S_{l_1,l_2}$ is removed from $S_{l_1,l_2}^{(t)}$.

2. A window $w(l_1,l_2,i,j,u,v) \in \mathcal{V} - S_{l_1,l_2}$ is added to $S_{l_1,l_2}$ if it does not create an overlap.

3. A window $w(l_1,l_2,i,j,u,v) \in S_{l_1,l_2}$ is replaced by a window $w(l_1,l_2,i',j',u',v') \in \mathcal{V} - S_{l_1,l_2}$ that only overlaps with $w(l_1,l_2,i,j,u,v)$ in $S_{l_1,l_2}$.

Further, let $S_{l_1,l_2}$ be a neighbor of itself, and define $\mathcal{N}(S)$ as the set of neighbors of $S$.

We add a further constraint on what constitutes an overlap. Two adjacent windows of the form $w(l_1,l_2,i,j,u,v)$ and $w(l_1,l_2,i-u,j-v,u',v')$, i.e., without a gap between their intervals on either level, should structurally be considered a single bigger window, $w(l_1,l_2,i,j,u+u',v+v')$. Figure 7.1 shows an example. However, since the sum of the weights of the smaller windows is not equal to the weight of the larger window, this creates a conflict. We resolve this by extending the definition of overlap to include such windows. This affects the set of windows available to operations 2 and 3 of Definition 7.1.

The following lemma allows us to use neighboring structures to define the proposal distribution.

**Lemma 7.1.** $X \in \mathcal{N}(Y)$ if and only if $Y \in \mathcal{N}(X)$. In addition, a path exists between any two states via a sequence of operations.

**Proof.** Suppose removing a window $w$ from structure $X$ creates structure $Y$. Then adding $w$ to $Y$ creates $X$. One can consider ‘replacement’ as a two step process: remove window $w_1$ from $X$...
to create an intermediate structure $R$, then add window $w_2$ in $R$ to create $Y$. Since neither $w_1$ nor $w_2$ overlap with $R$ (by definition of replacement) the process can be reversed: drop window $w_2$ from $Y$ to create $R$, then add $w_1$ to $R$ to create $X$.

Additionally, it is possible to reach any state $Z$ from $X$ even if $Z \not\in \mathcal{N}(X)$ by first dropping all windows of $X$ and then sequentially adding windows of $Z$. It may be possible for a shorter path to exist via replacement, or if $X \cap Z \neq \emptyset$.

The proposal distribution $Q(S_{l_1,l_2}^{(t+1)} | S_{l_1,l_2}^{(t)} )$ can be defined as a \textit{uniform} distribution over $\mathcal{N}(S_{l_1,l_2}^{(t)})$. That is,

$$Q(S_{l_1,l_2}^{(t+1)} = Y | S_{l_1,l_2}^{(t)} = X) = \begin{cases} \frac{1}{|\mathcal{N}(X)|} & \text{if } Y \in \mathcal{N}(X) \\ 0 & \text{otherwise} \end{cases} \quad (7.3)$$

Note that by definition, $|\mathcal{N}(X)| \geq 1$, since $X$ is in its neighborhood. Clearly the chain defined by the proposal distribution is irreducible. A good proposal distribution should not have a very low acceptance rate, otherwise the Markov chain $\{S_{l_1,l_2}^{(t)}\}$ will remain stuck in the same state for too long (such a chain is said to be \textit{sticky}) [Mur12]. The uniform distribution has a high enough acceptance rate. Furthermore, by using a uniform distribution as the proposal distribution, we are able to exploit some of its properties to develop a fast data structure to sample a neighbor in each iteration of the Metropolis-Hastings algorithm. We will discuss this data structure in the next section.
7.3 Data Structure

The Metropolis-Hastings algorithm generates a long chain of samples, so it is apparent that the majority of the work done by the sampling algorithm involves repeatedly computing neighborhoods of states and randomly choosing one neighbor. This merits the development of a fast algorithm to randomly choose a neighboring state. Our query is of the following form: given a structure \( S \), return all neighbors of \( S \), or perhaps, randomly sample one neighbor of \( S \). We can create neighbors by adding a window, removing a window, or replacing a window with a new one that overlaps it. If we want to generate all neighbors that can be created by adding a window, we need to find out which windows do not overlap with any of the existing windows in \( S \). Suppose \( \text{nonoverlaps}(w) \) is the set of all windows that do not overlap window \( w \). Then the set of windows that can be added to \( S \) is

\[
\bigcap_{w_i \in S} \text{nonoverlaps}(w_i)
\]

There are many ways of computing set intersections. An approach we experimented with was to sort the windows in the sets \( \text{nonoverlaps}(w) \) according to some total order (for example the order the windows were in in the input), then compute intersection using the ‘merge’ step from Merge Sort. If \(|S| = k\), the ‘merge’ step would require time \( O(nk) \) where \( n = \max_{w_i \in S} |\text{nonoverlaps}(w_i)| \). From a complexity perspective, this approach should be reasonable, since \( n \) is usually small compared to the number of all windows in the input, and \( k \) is much smaller. However, the cost of comparing windows in the ‘merge’ step, as well as specifically generating all intersections makes the implementation quite slow. In addition to computing intersections, we also need to compute unions. For instance, given a set \( \text{overlaps}(w) \) for each window \( w \), we would want to compute

\[
\bigcup_{w_i \in S} \text{overlaps}(w_i)
\]
which is the set of potential windows that could replace a window in $S$, since according to Definition 7.1, a window can only be replaced by one that overlaps it. This set will have to be filtered later to remove those windows that cause overlaps with other windows already in $S$. Instead of generating these sets, we take an alternate approach, based on two key ideas.

Firstly, we exploit the fact that set operations such as union, intersection and difference can be implemented very efficiently via bit arrays. Suppose we have a universe of elements $U$, and we have assigned a unique index $i$ to each element $e_i$ in $U$ such that $0 \leq i < |U|$. The set $U$ can now be represented as a bit array, such that the $i^{th}$ bit corresponds to element $e_i$. If the word size on the underlying architecture is $B$ bits, the size of this array would be $\lceil \frac{|U|}{B} \rceil$ words. Now, to represent any set $S \subseteq U$, we will use an array of the same size, and set the $i^{th}$ bit to 1 for each $e_i \in S$, and the remaining bits to 0. The space complexity of storing set $S$ is $\Theta(|U|)$ instead of $\Theta(|S|)$. However, we can justify this increased storage requirement because set operations will be very fast now. Suppose we have two sets $S_1$ and $S_2$ stored in this way, and we wish to compute their intersection. Then we only have to compute the logical (bitwise) AND of corresponding words in the arrays of $S_1$ and $S_2$, with a total cost of $\lceil \frac{|U|}{B} \rceil$ logical operations. The efficiency is further compounded by the caching features of the processor architecture. That is, due to the spatial locality considerations of processor caches, when a word is retrieved from the RAM, it’s neighboring words are also stored in the cache. Suppose the number of bits that are read from the memory in to the cache is $L$; then the total number of memory accesses required to process a set is $O\left(\frac{|U|}{L}\right)$. Typically, $L$ is large enough to store a few words. Since the intersection operation reads adjacent words in the array in a sequence, the first read would be sufficient to cache most of the array, if not in entirety, for most instances of the problem. In all, the intersection operation is almost free of cost. Set unions and differences can be implemented similarly. To retrieve the resulting sets for the union, intersection or difference we can select those elements from $U$ whose corresponding bits are 1.

Our second idea follows from the observation that we do not need to explicitly generate all neighboring states, due to the following two facts.
Fact 1: The proposal probability of moving from one state to another is independent of the windows in the structure and/or their weight. The proposal probability depends only on the number of neighboring states (it is its inverse).

Fact 2: The sets of neighbors created by each of the operations (addition, removal, replacement) are pairwise disjoint. This follows simply from the fact that, given a structure $S$, adding a window to $S$ increases the size of the new structure by one, removing a window decreases the size by one, and replacing a window keeps the size constant. In addition, no two replacement operations will have the same result, since the new window has to overlap the previous.

Thus the actual neighboring structures do not require to be generated; instead we only need to count all the neighbors (i.e., the number of different operations that can be performed), and output one randomly chosen neighbor as the candidate state (say $Y$), by randomly selecting an operation. These two quantities are sufficient to compute the acceptance probability

$$
\alpha(Y|X) = \min \left\{ 1, \frac{|\mathcal{N}(X)|}{|\mathcal{N}(Y)|} \times \frac{\exp[w(Y)/RT]}{\exp[w(X)/RT]} \right\}
$$

where $X$ is the current state.

We now turn towards the implementation of the sampling step that employs these two ideas. Let $\mathcal{W}$ be the set of all windows in the original input, including windows with negative weights. Assume that each window in $\mathcal{W}$ can be indexed by an integer in $[0, |\mathcal{W}|)$ (for example, $w_i = \mathcal{W}[i]$ always refers to the same window). Similarly, let $\mathcal{W}_{l_1,l_2} \subseteq \mathcal{W}$ be the (indexable) set of windows of $\mathcal{W}$ which have the even interval on level $l_1$ and the odd interval on level $l_2$.

Before we proceed to the description of the algorithm that samples a neighbor uniformly, we revisit the idea of cycles in mixed graphs of structures. It was stated earlier in Definition 7.1 that when computing the neighbors of a structure $S$, windows which create an overlap in the new structure are not included in the candidates for addition/replacement. We can, in fact, remove these windows before before the Metropolis-Hastings algorithm begins for a particular pair of levels $(l_1,l_2)$, instead of removing them whenever sampling a neighbor. Essentially, we have to find all windows whose addition will create a cycle in the mixed graph of $\cup_{(l',r') \in G-(l_1,l_2)} S_{l',r'}$. 

This can be done efficiently by computing directed paths between any two vertices in the mixed graph, then removing all windows in $W_{l_1,l_2}$ that join the start and end of any path from the set of windows that are candidates for addition and replacement operations. Let $cycles$ be the set of all such windows that create a cycle.

Recall from our earlier discussion that we assume we have sets $overlaps(w)$ and $nonoverlaps(w)$ for each window $w \in W$. We precompute these sets at the beginning of the program, and store this information in two boolean matrices (represented via bit arrays) in our data structure. The precomputation step is discussed later. The two two-dimensional bit arrays are as follows:

- $overlaps$, such that $overlaps[i][j]$ is 1 if distinct windows $w_i$ and $w_j$ overlap, and 0 otherwise (in other words, row $i$ of $overlaps$ represents the set $overlaps(w_i)$).
- $nonoverlaps$, such that $nonoverlaps[i][j]$ is 1 if distinct windows $w_i$ and $w_j$ do not overlap, and 0 otherwise.

Note that $overlaps[i][i] = nonoverlaps[i][i] = 0$ for all $i$.

In addition, we have a bit array to store windows that create cycles. Before sampling structures in a level, $completes_cycle[i]$ is set to 1 if the window $w_i$ is found to create cycles and 0 otherwise.

To sample a neighbor of structure $S$, we need the following sets. Let $A$ be the set of windows that can be added to the existing set $S$, $R$ be the set of windows that can be added by replacing a window in $S$, and $D$ be the set of windows that can be removed from $S$. Note that $A, R, D$ are pairwise disjoint.

$$A = \bigcap_{w_k \in S} nonoverlaps(w_k) - cycles$$  \hspace{1cm} (7.5)

$$R = \bigcup_{w_k \in S} \left\{ overlaps(w_k) \cap \left\{ \bigcap_{w_i \in S - w_k} nonoverlaps(w_i) \right\} \right\} - cycles$$  \hspace{1cm} (7.6)

$$D = S$$  \hspace{1cm} (7.7)

We have the following two operations.
Algorithm 7.2: next_action($S$)

Output: Pair $(action, window)$, and the size of the neighborhood $n$

1. $n \leftarrow 0$
2. // Add a window
3. foreach $w_i \in W_1, l_2$ do
4.    $x \leftarrow \neg \text{completes\_cycle}[i]$
5.    foreach $w_j \in S$ do
6.        $x \leftarrow x \land \text{nonoverlaps}[j][i]$
7.    end
8.    if $x = 1$ then
9.        $n \leftarrow n + 1$
10.       $r \leftarrow$ random integer in $[0, n)$
11.      if $r = 0$ then
12.        $(action, win) \leftarrow (ADD, i)$
13.      end
14.    end
15. end
16. // Replace a window
17. foreach $w_k \in S_1, l_2$ do
18.    foreach $w_i \in W_1, l_2$ do
19.        $x \leftarrow \text{overlaps}[k][i] \land \neg \text{completes\_cycle}[i]$
20.        foreach $w_j \in S - \{w_k\}$ do // Check overlaps with all windows in $S$
21.            $x \leftarrow x \land \text{nonoverlaps}[j][i]$
22.        end
23.    if $x = 1$ then
24.        $n \leftarrow n + 1$
25.        $r \leftarrow$ random integer in $[0, n)$
26.        if $r = 0$ then
27.            $(action, win) \leftarrow (REPLACE, (k, i))$
28.        end
29.    end
30. end
31. // Delete a window
32. foreach $w_k \in S_1, l_2$ do
33.    $n \leftarrow n + 1$
34.    $r \leftarrow$ random integer in $[0, n)$
35.    if $r = 0$ then
36.        $(action, win) \leftarrow (DELETE, k)$
37.    end
38. end
39. // Stay
40. $n \leftarrow n + 1$
41. $r \leftarrow$ random integer in $[0, n)$
42. if $r = 0$ then
43.    $(action, win) \leftarrow (STAY, \emptyset)$
44. end
• **next_action(S)**

This returns a window chosen uniformly at random from $A \cup R \cup D - cycles$ as well as the corresponding ‘action’ (i.e., addition/replacement/removal). The algorithm (Algorithm 7.2) works in three phases: first it considers the set $A$, then set $R$, then $D$, as well as considering the case where the candidate state is the current state. We maintain a candidate window $win$ throughout all phases. In the first phase, it considers each window in the set $W_{l_1,l_2}$ as a potential candidate, then determines whether it is a member of $A$, and if it is, replaces $win$ by this candidate with the probability $1/(n+1)$, or leaves $win$ unchanged with probability $1 - 1/(n+1)$, where $n$ is the number of members of $A$ already considered in the previous iterations (so initially $n = 0$). This method of randomly selecting an element of a set is known as *reservoir sampling* (also called *online sampling*), and it guarantees that after $i$ iterations through a list $L$, an element has been chosen uniformly at random from $L[1, \ldots, i]$. At the end of the first phase, $win$ is a window chosen uniformly at random from $A$.

Candidates from $R$ and $D$ are chosen similarly. The value of $n$ is not reset to 0 before phases 2 and 3, so that when phase two starts $n = |A|$, and at the beginning of phase three $n = |A| + |R|$. At the end, we choose the current state with probability $1/(|A| + |R| + |D| + 1)$.

• **get_nbr_size(S)**

This returns $|A| + |R| + |D|$, which is the value of $n$ after Algorithm 7.2 has terminated. Note that when computing the probability of going back from the candidate state to the current state (the term $Q(\sigma(i)|\sigma^C)$ in Eq. 7.2), we do not require sampling any state; we only need the size of the neighborhood. The pseudorandom number generation step can thus be avoided.

The key ideas in Algorithm 7.2 are that we can associate only individual windows with the next state, quickly detect whether a given window can create a neighboring state, and keep a random candidate state as we generate “next states” on the fly. The actual implementation of Algorithm 7.2 is slightly different in the sense that it performs the bitwise AND, OR and
COMPLEMENT operations on entire words instead of individual bits. Only lines 9 and 24 require consideration of each bit in a word. For very large sets, one can consider using approximate reservoir sampling on words; however, our implementation was fast enough to not require this technique, despite the fact that we generate pseudorandom numbers several times in the algorithm.

Construction of Overlap Matrices

To introduce our approach to creating the overlap matrices, we start with the case of just two levels, one even \((l_1)\) and one odd \((l_2)\). For each window \(w(l_1, l_2, i, j, u, v)\), we want to find all other windows that it overlaps, and set a bit correspondingly in both overlaps and nonoverlaps. The complexity of filling out the matrices is of course \(\Theta(|W|^2)\), where \(|W|\) is the number of windows in the input. However, as the cost (in terms of CPU cycles) of comparing two windows to determine whether they overlap or not far outweighs that of setting a bit, we have developed a method that avoids \(\Theta(|W|^2)\) comparisons. In fact, the method does not explicitly compare any two windows. In addition, if we maintain a set of windows in the implementation and use it compute the overlap matrix, we require \(O(|W|^2)\) memory accesses. Using bit arrays, we require only \(O\left(\frac{|W|^2}{T}\right)\) memory accesses.

A window that does not overlap \(w(l_1, l_2, i, j, u, v)\) appears either completely to its right, or completely to its left. Earlier we noted that a window \(w(l_1, l_2, i, j, u, v)\) is defined by a pair of intervals, \(l_1[i - u + 1, i]\) (the even interval) and \(l_2[j - v + 1, j]\) (the odd interval), so a non overlapping window appears to the left of both intervals, or to the right of both. In other words, a window \(w'(l_1, l_2, i', j', u', v')\) does not overlap \(w(l_1, l_2, i, j, u, v)\) if and only if \(i' < i - u + 1\) and \(j' < j - v + 1\), or \(i < i' - u' + 1\) and \(j < j' - v' + 1\). Let \(A\) be the set of windows whose even interval appears on the left of \(l_1[i - u + 1, i]\), and \(B\) be the set of windows whose odd interval appears on the left of \(l_2[j - v + 1, j]\). Then \(A \cap B\) is the set of windows which appear entirely on the left of \(w\). Similarly, define \(C\) to be the set of windows whose even interval appears on the right of \(l_1[i - u + 1, i]\), and \(D\) to be the set of windows whose odd interval appears on the right of \(l_2[j - v + 1, j]\). Then \(C \cap D\) is the set of windows which appear entirely on the right of \(w\).
The set of windows which do not overlap \(w(l_1, l_2, i, j, u, v)\), then, is the union \((A \cap B) \cup (C \cap D)\). Our goal is to efficiently compute these sets for all windows. For now, we only concern ourselves with overlaps in \((l_1, l_2)\), and defer more general overlap detection until later.

To efficiently compute the above mentioned set for all windows between \(l_1\) and \(l_2\), we first define the following sets for all pegs \(k\) on levels \(l_1\) and \(l_2\):

- \(R_{l_1}[k] = \) set of all windows whose even interval \(l_1[p, q]\) is on the right of \(k\), i.e., \(p > k\)
- \(R_{l_2}[k] = \) set of all windows whose odd interval \(l_2[p, q]\) is on the right of \(k\), i.e., \(p > k\)
- \(L_{l_1}[k] = \) set of all windows whose even interval \(l_1[p, q]\) is on the left of \(k\), i.e., \(q < k\)
- \(L_{l_2}[k] = \) set of all windows whose odd interval \(l_2[p, q]\) is on the left of \(k\), i.e., \(q < k\)

Assuming these sets have been precomputed for all possible \(k\), we extend the above idea (i.e., \((A \cap B) \cup (C \cap D)\)) for any given window \(w(l_1, l_2, i, j, u, v)\):

\[
\text{nonoverlaps}(w(l_1, l_2, i, j, u, v)) = (L_{l_1}[i - u] \cap L_{l_2}[j - v]) \cup (R_{l_1}[i] \cap R_{l_2}[j]) \quad (7.8)
\]

Since set unions and intersections can be computed very quickly using bit arrays, the problem reduces to finding the sets \(R_{l_1}[k], R_{l_2}[k], L_{l_1}[k], L_{l_2}[k]\) for each peg \(k\). We observe that:

- \(L_{l_1}[k] \supseteq L_{l_1}[k - 1]\)
- \(R_{l_1}[k] \supseteq R_{l_1}[k + 1]\)

and similar relations hold for \(l_2\). The only additional windows needed to compute \(L_{l_1}[k]\) (resp. \(R_{l_1}[k]\)) are those whose rightmost (resp. leftmost) even peg is \(k - 1\) (resp. \(k + 1\)). Let \(\overline{L}_{l_1}(k)\) be the set of windows whose leftmost even peg is \(k\), and let \(\overline{R}_{l_1}(k)\) be the set of windows whose rightmost even peg is \(k\). These can be found in \(O(|\mathcal{W}|)\) time for all possible pegs by scanning the windows once. Therefore,

\[
L_{l_1}[k] = L_{l_1}[k - 1] \cup \overline{R}_{l_1}(k - 1) \quad (7.9)
\]
\[
R_{l_1}[k] = R_{l_1}[k + 1] \cup \overline{L}_{l_1}(k + 1) \quad (7.10)
\]
We define similar recurrences for $l_2$. These can be efficiently computed with bottom-up dynamic programming. Assuming the length of the RNAs is $O(n)$, the total number of memory accesses required to compute $R_{l_1}[k], R_{l_2}[k], L_{l_1}[k], L_{l_2}[k]$ for all $k$ is $O\left(\frac{n|W|}{L}\right)$.

We turn our attention back to Eq. 7.11 and its shortcoming. Our definition of “overlap” precludes gaps of length zero on both levels (as stated earlier in Section 7.2), but this equation permits them. One strategy to take this into account could be to subtract, from the right hand side in Eq. 7.11, the set of all windows of the form $w(l_1, l_2, i - u - 1, j - v - 1, u''', v''')$ and $w(l_1, l_2, i + u' + 1, j + v' + 1, u', v')$ for all values of $u', v', u'', v''$, i.e., the set of all windows that “touch” $w(l_1, l_2, i, j, u, v)$. However, it will require computing this set first, which is not entirely desirable, so we take the following approach. Consider the set $L_{l_1}[k_1 - 1] \cap L_{l_2}[k_2]$. This is the set of all windows that have their even rightmost peg before $k_1 - 1$ and odd rightmost peg before $k_2$. Similarly $L_{l_1}[k_1] \cap L_{l_2}[k_2 - 1]$ is the set of all windows that have their even rightmost peg before $k_1$ and odd rightmost peg before $k_2 - 1$. Their union is the set of all windows to the left of pegs $k_1$ and $k_2$ that do not touch both $k_1$ and $k_2$. We can now redefine eq. 7.11 using this definition:

$$nonoverlaps(w(l_1, l_2, i, j, u, v)) = (L_{l_1}[i - u] \cap L_{l_2}[j - v + 1])$$
$$\cup (L_{l_1}[i - u + 1] \cap L_{l_2}[j - v])$$
$$\cup (R_{l_1}[i + 1] \cap R_{l_2}[j])$$
$$\cup (R_{l_1}[i] \cap R_{l_2}[j + 1]) \quad (7.11)$$

We can now generalize this algorithm to multiple levels. When determining whether there is an overlap between window $w(l_1, l_2, i, j, u, v)$ and another window $w'(l'_1, l'_2, i', j', u', v')$, there are four cases vis-a-vis which levels the second window lies in. If $l'_1 \neq l_1$ and $l'_2 \neq l_2$, then there cannot be an overlap between $w$ and $w'$. If $l'_1 = l_1$ and $l'_2 = l_2$, then this case can be handled by the algorithm presented previously. The remaining two cases are symmetric.

First, we will discuss handling the case where $l'_1 = l_1$ but $l'_2 \neq l_2$. Since $l'_2 \neq l_2$, we only concern ourselves with nonoverlapping intervals on $l_1$. The set of all such windows with intervals
that satisfy this constraint is \( L_{l_1}[i - u + 1] \cup R_{l_1}[i] \). However, this set also contains windows which have odd intervals on \( l_2 \). Earlier, we defined the set \( \mathcal{W}_{l_1,l_2} \) as the set of all windows whose even interval is on \( l_1 \) and odd interval is on \( l_2 \). Therefore, \( L_{l_1}[i - u + 1] \cup R_{l_1}[i] - \mathcal{W}_{l_1,l_2} \) results in the desired set of windows. Since the other case, where \( l'_2 = l_2 \) but \( l'_1 \neq l_1 \), is symmetric, the set of nonoverlapping windows is \( L_{l_2}[j - v + 1] \cup R_{l_2}[j] - \mathcal{W}_{l_1,l_2} \). Combining all four cases, we can claim that the set of all windows, from all levels, that do not overlap \( w(l_1, l_2, i, j, u, v) \) is

\[
\text{nonoverlaps}(w(l_1, l_2, i, j, u, v)) = (L_{l_1}[i - u] \cap L_{l_2}[j - v + 1]) \\
\quad \cup (L_{l_1}[i - u + 1] \cap L_{l_2}[j - v]) \\
\quad \cup (R_{l_1}[i + 1] \cap R_{l_2}[j]) \\
\quad \cup (R_{l_1}[i] \cap R_{l_2}[j + 1]) \\
\quad \cup (L_{l_1}[i - u + 1] \cup R_{l_1}[i] \cup L_{l_2}[j - v + 1] \cup R_{l_2}[j] - \mathcal{W}) \\
\quad \cup_{l'_1 \neq l_1, l'_2 \neq l_2} \mathcal{W}_{l'_1, l'_2} 
\] (7.12)

where the last line represents the set of windows which have neither interval in either \( l_1 \) or \( l_2 \), since these do not cause an overlap. Despite the apparent complexity of Eq. 7.12, the computation of \( \text{nonoverlaps}(\cdot) \) is incredibly fast, since the only operations required here are set union, intersection and difference, and each of these can be implemented very efficiently, as we have already discussed. Note that in the implementation, \( \text{nonoverlaps}(\cdot) \) is a bit array, so it can immediately be set as a row in matrix \( \text{nonoverlaps} \). The row in \( \text{overlaps} \) is simply the complement of \( \text{nonoverlaps}(\cdot) \), except for the case of \( \text{overlaps}[i][i] \), which is always 0.

Figure 7.2 shows a comparison the time requirements of our approach versus a naive approach that performs pairwise comparison to compute overlaps, for the case of two levels. In both instances, the time required is dependent on the number of windows in the input, rather than the length of the RNAs. To perform meaningful comparison, we randomly generate windows for different level lengths. Suppose the length of both levels is fixed at \( n \). We then generate all possible windows of the form \( w(l_1, l_2, i, j, u, u) \), i.e., with side lengths equal, where \( 1 \leq u \leq 15 \).
(a) The times took by both approaches to create the matrices, as a function of the length of each RNA.

(b) The ratio of times, i.e., \( \frac{\text{pairwise comparison}}{\text{set based ops}} \), as a function of the length of each RNA.

Figure 7.2: Performance of the set based data structure compared to a naive pairwise comparison based approach, during the creation of overlaps and nonoverlaps matrices.

At \( n = 100 \), with approximately 120,000 windows, the set based approach requires only 4.7 seconds, as opposed to the naive approach which requires about 200 seconds.
7.4 Experimental Results

Even though sampling algorithms tend to produce a large spectrum of structures, many of the results have very similar shapes. We can use the distance function introduced in Section 6.3 with hierarchical clustering to cluster the sampling output into classes. However, the distance function’s (Eq. 6.3) precondition is that the structures should have been produced by Algorithm 6.1, since it depends on the order of windows, whereas the sampling algorithm adds windows without regards to order. This can be resolved easily. Windows in any valid structure have a total order defined on them via terminal windows, even if they are not in that order in a given representation (in this case, the set created by the sampling algorithm). An $O(|S|^2)$ procedure to sort a set of windows $S$ is as follows:

1. Maintain an ordered sequence $L$, initially empty.
2. Find the terminal window $w^T$ in $S$, and add it to $L$.
3. Remove $w_i$ from $S$ and goto step 2 if $S \neq \emptyset$.
4. Reverse $L$ and return.

To prepare for clustering, we first remove duplicates, so we only work with unique samples, and compute pairwise distances of the remaining samples. We then use hierarchical agglomerative clustering with complete linkage, and obtain the clusters by “cutting” the tree where distance between clusters is 1 (largest). Figure 7.3 shows an example of a dendrogram that results from applying hierarchical clustering on the output of the sampling algorithm for yeast U6, U2 and introns. The algorithm outputs 1000 samples, but not all of them may be unique. For this particular input, only 214 samples out of 1000 are unique, since the number of windows in the input is small. The agglomerative hierarchical clustering algorithm iteratively merges two clusters at a time (to create a single cluster), so the dendrogram is actually a binary tree, representing which two clusters were merged. The height of an internal node in the tree represents the distance
Figure 7.3: A dendrogram visualizing the result of hierarchical clustering performed on 214 unique samples of the U6-U2 and introns complex (the total number of samples was 1000). We are able to identify nine clusters which have a distance of 1 from each other. Leaf nodes representing elements in the same cluster have the same color.

between its two clusters.\(^1\) By “cutting” the tree at height 1, we are able to identify clusters which have the maximum distance between them.

Given the clusters, the optimal solution in each cluster acts as a “representative” of the cluster. The representatives should reveal some suboptimal structures that are biologically correct [AM14, MA15, MA16]. We sort the representatives of the clusters by decreasing weight. We consider the first \(k\) representatives, for a given \(k\). To assess our approach, we repeat the experiment 200 times, which was verified to be enough for the percentage hits (defined below) to converge within ±3%. Given a set of candidate structures in mind; for instance, Figure 7.4 shows four candidates for the yeast spliceosome, we then count for each candidate the number of runs in which it is found among the first \(k\) representatives, as a percentage hit. We also compute the “rank” of each candidate, which is the index of it’s first \(^2\) representative (according to the sorted order) if found, averaged over the number of runs with a hit. Finally, we compute for each candidate the \(F_1\)-score of its first representative when found, also averaged in the same way. After converting windows to base pairs, the \(F_1\)-score is given by

\[
F_1\text{-score} = 2 \times \frac{\text{precision} \times \text{recall}}{\text{precision} + \text{recall}}
\]

\(^1\) It may appear that in Figure 7.3, the root has nine subtrees instead of two; however, that’s a side effect of the visualization of the dendrogram, since the distance between the nine subtrees is the same.

\(^2\) We use “first representative” because many solutions can represent the same candidate; for instance, a window in CopA-CopT can split in different ways, but we still refer to it as a window split.
7.4. Experimental Results

Figure 7.4: The yeast spliceosome with 4 RNAs (I1 and I2 are functionally independent stretches of the same much longer messenger RNA). (a) Helix Ia and helix Ib with both introns attached. (b) Helix Ia and helix Ib with I1 detached. (c) Helix Ia with both introns attached. (d) Helix Ia with I1 detached. Both (a) and (c) represent biologically correct structures.

where \textit{recall} is defined as the number of base pairs in the representative that are also in the biologically correct structure, divided by the total number of base pairs in the latter, and \textit{precision} is defined as the same but divided by the total number of base pairs in the representative.

We consider three settings with (a) all windows included (no filtering), (b) only symmetric windows \( w(l_1,l_2,i,j,u,v) \) where \( u = v \), and finally (c) the “best” 500 windows \( w(l_1,l_2,i,j,u,v) \) based on highest \( Z_{l_1,l_2}^I (i - u + 1, i, j - v + 1, j) \) among all bounded size windows satisfying \( \max(u,v) \leq 10 \).

7.4.1 Structural Variation

The interaction of the U2-U6 complex in the spliceosome of yeast (shown in Figure 3.10) has the pattern I1-U6-U2-I2 (the bipartite interaction graph is a path). The complex has been reported

<table>
<thead>
<tr>
<th>Class</th>
<th>Without Filtering</th>
<th>Symmetric Windows</th>
</tr>
</thead>
<tbody>
<tr>
<td>Helices Ia+Ib (2 symm. windows)</td>
<td>100 1 1</td>
<td>100 1 1</td>
</tr>
<tr>
<td>Helices Ia+Ib (2 symm. windows), I1 detached</td>
<td>100 2 0.914</td>
<td>100 2 0.914</td>
</tr>
<tr>
<td>Helix Ia</td>
<td>100 3 1</td>
<td>100 3 1</td>
</tr>
<tr>
<td>Helix Ia, I1 detached</td>
<td>100 4 0.897</td>
<td>100 4 0.897</td>
</tr>
</tbody>
</table>

Table 7.1: Results for the yeast spliceosome.
to have two distinct experimental structures, e.g. [SCB04]. In one conformation, U2 and U6 interact to form a helix known as helix Ia. In another conformation, the interaction reveals a structure containing an additional helix, known as helix Ib. Section 3.10 describes possible underlying mechanisms that are responsible for this conformational switch. We consider the set of four candidates in Figure 3.10. The results are summarized in Table 7.1; we did not consider bounded size windows here because all given windows are already small in size.

7.4.2 Artifact Interactions

The pairwise interaction of CopA-CopT (the bipartite interaction graph is simply an edge) is shown in Figure 3.10. As discussed in Chapter 3, due to the optimization nature of our problem, it is sometimes possible to pick up interactions that are not biologically real; dropping these interactions from the solution would make it sub-optimal, even when preferred biologically. The last interaction window of CopA-CopT in Figure 3.13c is an example of such an artifact.

For each of the three interaction windows in Figure 3.13c, we consider whether the window is present, dropped, or split. We therefore identify six classes of candidates based on presence/absence of windows and window splits, as shown in Table 7.2.

A window $w(l_1, l_2, i, j, u, v)$ is considered present if the solution contains a window $w(l_1, l_2, i', j', u', v')$ “in range” such that $[i' - u' + 1, i'] \subseteq [i - u + 1 - 3, i + 3]$ and $[j' - v' + 1, j'] \subseteq [j - v + 1 - 3, j + 3]$. Furthermore, if exactly two windows fall in that range, we consider them as a window split. Typically, though we do not enforce it, such a window split is declared when the two windows happen to be treated as dependent.

The solution with the highest F1-score is characterized by a first window, and a middle window split, with the last window dropped (biologically correct). This solution is revealed almost always with as few as two clusters ($k = 2$), and, as shown in Table 7.2, has a relatively small rank ($\leq 2$) when $k = 5$. This is primarily attributed to the use of bounded size windows. However, even with a less stringent filtering of windows (“All” or “Symmetric”), the solution still shows up when the number of clusters $k$ is high enough (see also Footnote 3).

\footnote{Since a single non-symmetric window may also represent a split, our percentage hit for window splits is lower than it should be with the no filtering option.}
7.4. Experimental Results

<table>
<thead>
<tr>
<th>Class</th>
<th>Without Filtering</th>
<th>Symmetric Windows</th>
<th>Bounded Size Win.</th>
</tr>
</thead>
<tbody>
<tr>
<td>First, middle, last</td>
<td>92.9</td>
<td>99.3</td>
<td>8.5</td>
</tr>
<tr>
<td>First, middle split, last</td>
<td>7.6</td>
<td>88</td>
<td>100</td>
</tr>
<tr>
<td>First, middle, last dropped</td>
<td>87.9</td>
<td>97.7</td>
<td>73.2</td>
</tr>
<tr>
<td>First, middle split, last</td>
<td>7.1</td>
<td>37</td>
<td>100</td>
</tr>
<tr>
<td>First split, middle, last</td>
<td>2.7</td>
<td>61.7</td>
<td>0</td>
</tr>
<tr>
<td>First split, middle, last</td>
<td>1.8</td>
<td>20</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 7.2: Results for CopA-CopT, percentage hit followed by rank and F1-score.

<table>
<thead>
<tr>
<th>Class</th>
<th>Without filtering</th>
<th>Symmetric Windows</th>
<th>Bounded Size Windows</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1 (H1 as non-symm. window)</td>
<td>100</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>T2 (H1 &amp; H2 as 1 symm. wins.)</td>
<td>15.6</td>
<td>100</td>
<td>0</td>
</tr>
<tr>
<td>T3 (H1 &amp; H2as2symm. wins.)</td>
<td>79.7</td>
<td>100</td>
<td>100</td>
</tr>
</tbody>
</table>

Table 7.3: Results for the 4-way junction construct in the hairpin ribozyme complex. T3 is the correct structure.

7.4.3 Circular Interactions

We now revisit the 4-way junction construct of a hairpin ribozyme complex from Chapter 4 (Figure 3.11). We distinguish between three types of solutions: Type 1 is the “optimal” where H1 is predicted as a non-symmetric windows, adding a C-G pair and a U-A pair (thus a bulge in H1 on the RzA side). These additional pairs unwind in the actual structure possibly due high order interactions with other parts of the structure [PHH⁺01]. Type 2 (the second optimal) is when H1 and H2 are predicted as one symmetric window that extends from the 5’ end of the Substrate (and the 3’ end of RzA) to the center of the 4-way junction (thus with the same additional pairs reported above). Type 3 is the correct structure as shown in Figure 3.11. We used the same ±3 criteria for window boundaries as described in Section 7.4.2. When symmetric windows are considered, Type 3 will present itself as the second representative (suboptimal). The results are summarized in Table 7.3.

7.4.4 Star Interactions

We also predict suboptimal solutions for the three other examples of ribozyme complexes discussed in Chapter 4. Recall that one RNA interacts with the rest of the RNAs, thus forming a star bipartite interaction graph (Figure 4.2). For the hammerhead ribozyme complex, the
### Table 7.4: Results for the hammerhead ribozyme complex, percentage hit followed by rank and F1-score.

<table>
<thead>
<tr>
<th>Class</th>
<th>Without Filtering</th>
<th>Symmetric Windows</th>
<th>Bounded Size Windows</th>
</tr>
</thead>
<tbody>
<tr>
<td>Actual structure</td>
<td>100 1 0.999</td>
<td>100 1 1</td>
<td>100 1 0.944</td>
</tr>
<tr>
<td>five windows</td>
<td>63.8 2.3 0.954</td>
<td>98.7 2 0.959</td>
<td>0 - -</td>
</tr>
<tr>
<td>three windows</td>
<td>9.4 5 0.892</td>
<td>0.4 5 0.882</td>
<td>0 - -</td>
</tr>
</tbody>
</table>

### Table 7.5: Results for the hairpin ribozyme HP-WT complex, percentage hit followed by rank and F1-score. T3 is the correct structure.

<table>
<thead>
<tr>
<th>Class</th>
<th>Without Filtering</th>
<th>Symmetric Windows</th>
<th>Bounded Size Windows</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1 (H1 as non-symm. window)</td>
<td>100 1 0.955</td>
<td>0 - -</td>
<td>100 1 0.955</td>
</tr>
<tr>
<td>T2 (H1 &amp; H2 as 1 symm. wins.)</td>
<td>0 - -</td>
<td>100 1 0.955</td>
<td>0 - -</td>
</tr>
<tr>
<td>T3 (H1 &amp; H2 as 2 symm. wins.)</td>
<td>96.4 5 1</td>
<td>100 2 1</td>
<td>100 4 1</td>
</tr>
</tbody>
</table>

### Table 7.6: Results for the hairpin ribozyme HP-RJ complex, percentage hit followed by rank and F1-score. T3 is the correct structure.

<table>
<thead>
<tr>
<th>Class</th>
<th>Without Filtering</th>
<th>Symmetric Windows</th>
<th>Bounded Size Windows</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1 (H3 non symm.)</td>
<td>100 2 0.905</td>
<td>0 - -</td>
<td>100 1 0.905</td>
</tr>
<tr>
<td>T2 (H3 and H4 as one window)</td>
<td>100 1 0.95</td>
<td>100 1 0.95</td>
<td>0 - -</td>
</tr>
<tr>
<td>T3 (H3 and H3 as two symm.)</td>
<td>0 - -</td>
<td>100 3 1</td>
<td>0 - -</td>
</tr>
</tbody>
</table>

The correct solution always shows up in the first cluster (see Table 7.4). For the HP-WT and HP-RJ ribozyme complexes, results are similar to those of the 4-way junction construct, and are shown in Tables 7.5 and 7.6, respectively.
Chapter 8

Conclusion

While RNA-RNA interaction algorithms exist, they are not suitable for predicting RNA structures in which more than two RNA molecules interact. For instance, the interaction pattern may not be known, in contrast to the case of two RNAs where one must interact with the other. Moreover, even with some existing knowledge on the pattern of interaction, treating the RNAs pairwise may not lead to the global optimal structure. Despite there being significant progress in the development of algorithms for prediction of interaction of two RNAs, algorithms for multiple RNAs predict a very limit set of structures. In our thesis, we have presented a framework to predict a wider class of multiple RNA interaction complexes. We formulated multiple RNA interaction as an optimization problem based on our framework Pegs and Rubber Bands, prove it is NP-complete, and provide approximation and heuristic algorithms. We then introduced a more general interaction model, Bipartite Pegs and Rubber Bands, which allowed us to define arbitrary interaction patterns between two classes of RNAs.

In RNA interaction, the “optimal” structure may not be the correct structure, and the correct structure is not necessarily unique. We extend the Bipartite Pegs and Rubber Bands formulation to generate multiple suboptimal solutions with dependent interactions. Our approach successfully computes suboptimal solutions for the multiple RNA interaction problem that are truthful representations of the actual biological structures. For instance, it can provide several candidate
structures when they exist, e.g. for the U2-U6 complex and its introns in the spliceosome of yeast; and identify structures that are biologically correct, but are not necessarily optimal in the computational sense, e.g. for CopA-CopT in E. Coli and several ribozyme complexes.

In the process of addressing the shortcomings described above, we have developed a new framework that opens up exciting new theoretical problems in the domains of optimization algorithms, approximation algorithms, and tractability, many of which we discussed in this thesis, along with their solutions.

We anticipate that this thesis will motivate some problems to be solved in the future. We have presented an optimization algorithm for the basic Pegs and Rubber Bands model with the complexity $O(mn^m)$. A reasonable question to ask is, can this complexity be improved? That is, is there an algorithm with complexity $O^*(n^{m-\epsilon})$ for some $\epsilon > 0$, where $O^*(.)$ hides any polynomial factors? Some recent work has shown that the LCS problem with $k$ strings over an alphabet of size $O(k)$ does not have a $O(n^{k-\epsilon})$ solution unless the Strongly Exponential Time Hypothesis is false [ABW15]. Perhaps similar results can be achieved for the Pegs and Rubber Bands optimization problem. In addition, the approximability of optimization problem of the Bipartite Pegs and Rubber Bands model is worth investigating as well. Can we achieve a polynomial time approximation algorithm for the optimization problem? We have presented approximation algorithms for the path and cycle interaction patterns, so can approximation algorithms also be designed for the optimization problem over specific interaction patterns, such as the star graph?

Pegs and Rubber Bands is a new addition to the family of combinatorial frameworks, and as such forms the basis of many interesting problems. We hope that this thesis has started the process of investigating and solving those problems.
Bibliography


