Linear time algorithm for parsing RNA secondary structure

by

Baharak Rastegari

B.Sc., Sharif University of Technology, Tehran, Iran, 2002

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF

THE REQUIREMENTS FOR THE DEGREE OF

Master of Science

in

THE FACULTY OF GRADUATE STUDIES

(Department of Computer Science)

We accept this thesis as conforming
to the required standard

The University of British Columbia

August 2004

Library Authorization

In presenting this thesis in partial fulfillment of the requirements for an advanced degree at the University of British Columbia, I agree that the Library shall make it freely available for reference and study. I further agree that permission for extensive copying of this thesis for scholarly purposes may be granted by the head of my department or by his or her representatives. It is understood that copying or publication of this thesis for financial gain shall not be allowed without my written permission.

BAHARAK RASTEGARI — Name of Author *(please print)*

17/08/2004 — Date (dd/mm/yyyy)

Title of Thesis: Linear time algorithm for parsing RNA secondary structure

Degree: Master of Science Year: 2004

Department of Computer Science
The University of British Columbia
Vancouver, BC Canada

Abstract

RNA secondary structure prediction is an important problem in computational molecular biology. Experiments show that existing polynomial time prediction algorithms have limited success in predicting correctly the base pairs, i.e. secondary structure, in known biological RNA structures. One limitation of many current algorithms is that they can predict only restricted classes of structures, excluding many so-called pseudoknotted secondary structures. The type of the pseudoknotted structures that occur in biological structures, as well as the type of structures handled by current algorithms, have been poorly understood, making it difficult to assess the generality of current algorithms.

In this thesis we present a comprehensive and precise classification of structural elements and loops in a secondary structure, along with a linear time algorithm for parsing secondary structures into their structural elements.

The parsing algorithm, along with the classification scheme for the loops in a pseudoknotted secondary structure, can be used in analysing existing prediction algorithms to determine which known biological RNA structures can not be predicted by the algorithms. This analysis can help us to design new and more powerful prediction algorithms.

Furthermore, we present two applications of our work: (i) linear time free energy calculation algorithm, and (ii) linear time test for Akutsu's[2] algorithm class.

We present a linear time algorithm for calculating the free energy of a given secondary structure. This algorithm can be useful especially in heuristic prediction algorithms, as they commonly use a procedure to calculate the free energy for a given sequence and structures.

We also present a linear time algorithm to test whether the prediction algorithm introduced by Akutsu[2] can handle a given structure. The result of our analysis on algorithm of Akutsu on some sets of biological structures shows that although it is proved theoretically that the class of structures handled by Akutsu's algorithm is more general than that handled by the algorithm of Dirks and Pierce[7], they can both handle the same class of given biological structures.

Contents

List of Tables

List of Figures

Acknowledgements

First of all, I would like to thank my supervisor Dr. Anne Condon, who has guided me through research in bioinformatics since I did not even know what it meant. I am deeply grateful to her encouragements , her support, great research ideas she shared with me and her great guidance throughout the whole work.

I would also like to thank Dr. Will Evans for being my second reader and for helpful comments. I want to thank the people in Beta Lab and other collaborators who helped me throughout, especially Mirela Andronescue who has always helped me with her helpful comments and suggestions.

I also want to thank my dearest friends in vancouver for understanding me during the last months of work and keeping me calm whenever I felt frustrated. Last but not the least, I want to thank my parents, my dear sister Tahmineh and my dear brother Siavash for their love, support and trust.

BAHARAK RASTEGARI

The University of British Columbia
August 2004

To my dad and ma.

Chapter 1

Introduction

In this chapter, we first describe the research problem that we address in this thesis, parsing a given RNA secondary structure to its structural components, and give some motivations to support the importance of the work. Then in Section 1.2, we describe the contributions of our work in other applications such as calculating the free energy of a given RNA secondary structure. The last section is a brief overview of this thesis.

1.1 Parsing Secondary Structures: Problem and Motivation

An RNA molecule can be represented as a single stranded sequence consisting of the following four bases or nucleotides: Adenine (A), Guanine (G), Cytosine (C), and Uracil (U). This single stranded sequence is called *primary structure* or *sequence*.

RNA molecules play diverse roles in the cell: as carriers of information, catalysts in cellular processes, and mediators in determining the expression level of genes [6]. The molecule folds into a functional shape (structure) by forming intramolecular base pairs among some of its bases. These pairings arise from the Hydrogen-bonding forces between pairs of bases. The *secondary structure* describes which bases of the molecule bond with each other. Figure 1.2(a) and Figure 1.3(a) illustrate the base pairs in two simple secondary structures. The base pairs together with unpaired bases in these examples form the following structural components: stacked pairs (stems), hairpin loops, bulge loops, internal loops, multiloops, and pseudoknots (Figure 1.1). Then the secondary structure can also be represented by a list of structural components. More detailed definition is given in Chapter 2.

The structure of a functional RNA molecule is often the key to its function and interaction with other molecules, since it is conserved across many organisms though the sequence may vary. Therefore predicting the secondary structure of RNA molecules (or being more general, nucleic acids) is one of the most important problems in computational molecular biology.

Comparative sequence analysis is the most reliable approach for secondary structure prediction but it needs several sequences available. When just a single molecule is available,

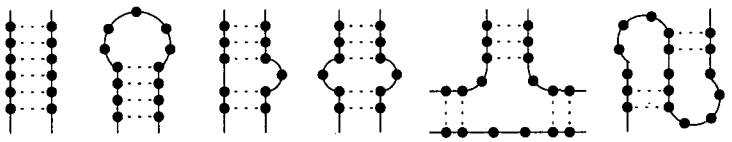
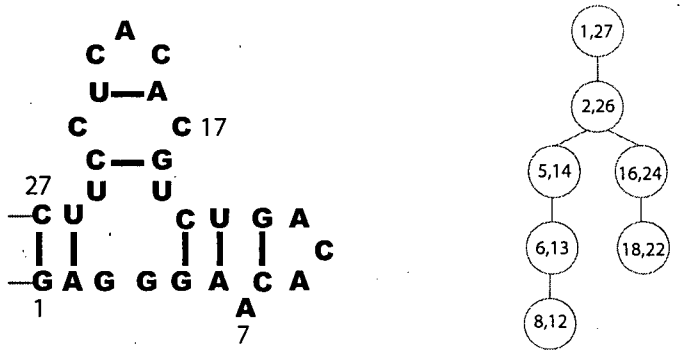


Figure 1.1: RNA secondary structural components (left-right): stem, hairpin loop, bulge loop, internal loop, multi-branch loop, pseudoknot.

computational prediction of its secondary structure from its base sequence is based on the premise that out of exponentially many possibilities, an RNA molecule is most likely to fold into the *minimum free energy (mfe)* structure. The free energy of a given structure is estimated by summing thermodynamic and entropic free energy terms associated with the component loops of the secondary structure.

Unfortunately, finding the mfe secondary structure for a given RNA sequence is NP-hard [11]. Several polynomial time algorithms have been proposed for predicting the mfe secondary structure from restricted classes of secondary structure. The most well known such class is that of pseudoknot free secondary structures and the work on prediction of secondary structure has mostly focused on prediction of pseudoknot free secondary structure. In Figure 1.2(a) a pseudoknot free structure is shown. Many biological RNA structures are pseudoknot free but there are also many natural structures which are pseudoknotted. Experiments showed that existing polynomial time algorithms are capable of predicting at most 70% of base pairs correctly in known (experimentally determined) biological RNA structures.



(a) (b)

Figure 1.2: (a) A pseudoknot free structure. (b) The tree of loops for the secondary structure given in part (a).

Heuristic algorithms provide one widely used approach for predicting secondary

structures. These algorithms commonly need a procedure to calculate the free energy for a given sequence and structure. So it is useful here to be able to compute what are the loops of the structure. If the structure is pseudoknot free, loops are either nested in each other, or disjoint, so it is natural to represent the loops and their relationships as a tree (Figure 1.2(b)). Although it was not reported before, it is straightforward to construct the tree from the description of the pseudoknot free secondary structure in linear time, if the structure is represented in standard format (see Chapter 2). Construction of the tree from the description of the secondary structure is what we mean by parsing the structure.

No previous work has provided a comprehensive way to classify the types of loops that arise in pseudoknotted structures, although certain types of pseudoknotted structures have been named. Figure 1.3(a) shows a simple pseudoknotted structure.

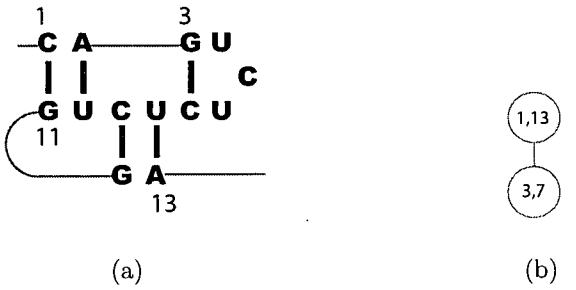


Figure 1.3: (a) A pseudoknotted structure. (b) The tree of loops for the secondary structure given in part (a).

Although there are algorithms for finding the minimum free energy secondary structure from limited classes of pseudoknotted structures, the papers describing these algorithms only implicitly (via description of their algorithm) explain what is their energy model, and what class of structures their algorithm can handle. While our long-term goal is to develop better algorithms for predicting pseudoknotted secondary structures, we felt that the first step would be to elucidate what is the energy model used by current algorithms. In turn, doing this means that a categorization of the types of loops which arise in pseudoknotted secondary structures is needed, and an efficient algorithm for parsing secondary structures into their component loops (Figure 1.3(b)) is useful. It should be noted here that the simple example in Figure 1.3(a) does not include all the different types of pseudoknotted loops that can arise in a real secondary structure (see Chapter 2).

1.2 Contributions

In this thesis we present a comprehensive way to classify the type of loops that arise in a pseudoknotted structure, and also an algorithm which parses a given secondary structure to

its structural components. Our parsing algorithm can be used in analysing and understanding known experimental structures (in order to design more efficient prediction algorithms), efficient free energy calculation of a secondary structure and other applications.

The first two contributions of our work are:

- A classification scheme for the loops in a pseudoknotted secondary structure (presented in Chapter 2), and
- A linear time parsing algorithm for constructing the parse tree of a given secondary structure (presented in Chapter 4).

The linear time algorithm for parsing a secondary structure can be used in several applications. In this work we present two of them as the next two contributions of our work:

- A linear time algorithm to calculating the free energy of a given secondary structure.

  Standard thermodynamic functions calculate the free energy of each loop using information about the loop such as: (i) its external base pair, (ii) an ordered list of its base pairs or tuples, (iii) number of unpaired bases in the loop, and (iv) type and the location of the loop. The free energy of a secondary structure is calculated by summing the free energy of its component loops. So it could be easily calculated in linear time if we have all loops and needed information. We use our algorithm for parsing the secondary structure to identify all loops in the structure, plus the needed information about them, in linear time. Therefore it is possible to calculate the free energy of a given secondary structure in linear time.

- A linear time algorithm to test whether the prediction algorithm introduced by Akutsu [2] can handle a given structure.

  There are polynomial time algorithms which have been designed to predict the secondary structure for RNA strands. As the problem is NP-hard, each of them can handle a restricted class of structures. It is useful to have a characterization of the classes of structures handled by each of these algorithms. Moreover, it is good to know whether the class of structures handled by an algorithm includes most known biological structures. Condon et al.[5] provide a simple characterization of the classes of structures handled by Rivas and Eddy (R&E) [13], Dirks and Pierce (D&P) [7] and Lyngso and Pederson (L&P) [11] algorithms. They also provide a linear time algorithm to test if an input structure is in the R&E, D&P, and L&P classes. But they didn't provide a test algorithm for Akutsu class. Using the parsing algorithm, we introduce a linear time algorithm to test whether an input structure is in the Akutsu class. Then we apply it on the same structures used by Condon et al.[5], to identify the number of structures that can be handled by the Akutsu algorithm.

1.3 Overview

In Chapter 2, we provide definitions of structural elements and loops in RNA secondary structure, where some notions, such as closed regions, bands and etc, are introduced in this work for the first time. In Chapter 3, we describe the previous related works, mostly in predicting RNA secondary structure. In Chapter 4, we describe our linear time algorithm for parsing a given structure to its closed regions (as structural components). We also provide a precise proof as to why it works accurately and in linear time. We present an application of our parsing algorithm in calculating the free energy of a given secondary structure, in Chapter 5. In Chapter 6, we present another application, namely a linear time test algorithm for Akutsu class. We also apply Akutsu class test algorithm on the number of known biological structures and present the result in this chapter. In Chapter 7 we end with a conclusion of our work and possible future work.

Chapter 2

Components of a Pseudoknotted Secondary Structure

In this chapter we present a formal definition of RNA secondary structure, along with precise definition and classification of structural elements and loops in a secondary structure.

For the first time, in this work we introduce *closed region* and *bands* notions, in Section 2.2. In Section 2.3 we present a precise definition of different type of loops in a secondary structure, where types of loops in pseudoknotted structure such as *interior-pseudoknotted loop*, *multi-pseudoknotted loop* and *pseudoknotted loop* are introduced for the first time. In the last section, the notion of *parse tree* is introduced.

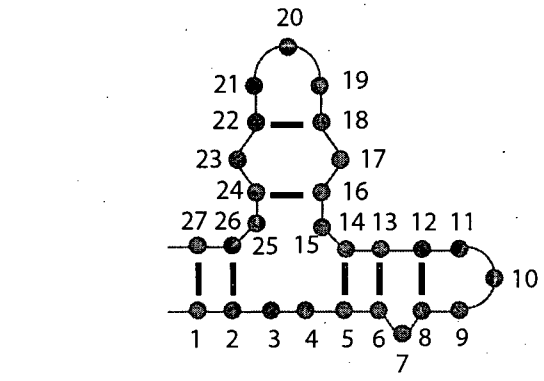
2.1 Components of a Pseudoknotted Secondary Structure

An RNA molecule can be represented as a single stranded sequence consisting of the following four bases or nucleotides: Adenine (A), Guanine (G), Cytosine (C), and Uracil (U). This single stranded sequence is called *primary structure* or *sequence*. The sequence has distinct $5'$ and $3'$ ends.

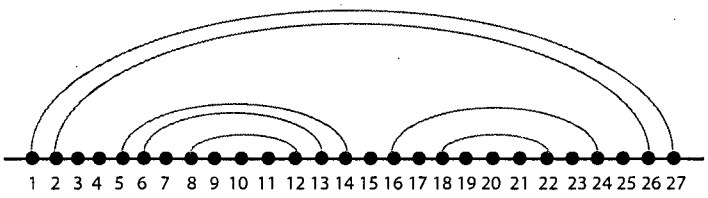
The molecule folds into a functional shape by forming intramolecular base pairs among some of its bases. These pairings arise from the Hydrogen-bonding forces between pairs of bases. The set of these base pairings is known as the secondary structure of the RNA. The Watson-Crick base pairs: {AU,CG,GC,UA} and wobble pairs: {GU,UG} are the set of common base pairings.

Formally, a set R of base pairs is called a *secondary structure* if each base is in at most one base pair. We index the bases consecutively from the $5'$ end toward the $3'$ end, starting at 1. Let $i.j$ $(i < j)$ denote that the base indexed i is paired with the base indexed j, in which case we write $i.j \in R$. For convenience, we write $i.j$ to mean $i.j \in R$. We refer to the length of the sequence by n (Figure 2.1(a) and Figure 2.2(a)).

There are different ways for representing RNA secondary structure. Figure 2.1(a) shows an example of usual representation of RNA secondary structure. But to easily explain



(a)



(b)

Figure 2.1: (a) Pseudoknot free secondary structure of length 27. (b) Arc diagram representation of the structure in part (a)

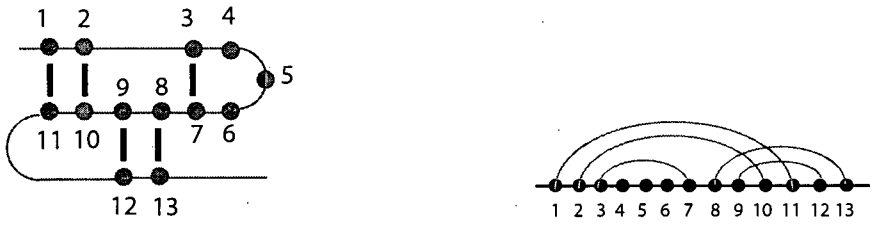
how algorithms work, it is usually easier to use other representations. In this work, we use *arc diagrams* to represent secondary structures. In the arc diagram representation of structure R, base indices are shown as vertices on a straight line (backbone), ordered from the $5'$ end, and arcs (always above the straight line) indicate base pairs (Figure 2.1(b) and Figure 2.2(b)).

We introduce a pattern representation of secondary structures as an alternative, which will be used in Chapter 6. In a pattern, information about unpaired bases and consequently the base indices is lost but the pattern of nesting or overlaps among base pairs is preserved. (We note that the definition of pattern could be extended so that unpaired bases are represented using a special symbol.) For example, the pattern representation of structure in Figure 2.1 is $P = abcdeedcfggfba$.

To define patterns precisely, we introduce some notation. We use ϵ to denote the empty string. Let N_n denote the natural numbers between 1 and n (inclusive). For any string s over alphabet Σ, $s \downarrow \sigma$ denotes the string s with all occurrences of σ removed, and $s \downarrow \Sigma'$, $\Sigma' \subset \Sigma$, denotes the string s with all occurrences of all $\sigma \in \Sigma'$ removed. Also $|s|$ denotes the size (length) of s.

Patterns: A string P (of even length) over some alphabet Σ is a secondary structure pattern, or simply a pattern, if every symbol of Σ occurs either exactly twice, or not at all, in P. We say that secondary structure R for a strand of length n corresponds to pattern P if there exists a mapping $m : N_n \rightarrow \Sigma \cup \{\epsilon\}$ with the following properties: (i) if $i.j \in R$ then $m(i) \in \Sigma$ and $m(i) = m(j)$, (ii) if $i.j$ and $j.i \notin R$ for all $j \in N_n$, then $m(i) = \epsilon$, and (iii) $P = m(1)m(2)...m(n)$.

We refer to the index of the first and the second occurrence of any symbol σ in P by $Left(P, \sigma)$ and $Right(P, \sigma)$ respectively. When P is understood, we use $Left(\sigma)$ and $Right(\sigma)$. In the same way, the first occurrence of any symbol is called a *Left symbol* where the second occurrence is called a *Right symbol.* For example, pattern $P = abcdeedcfggfba$ corresponds to the structure in Figure 2.1(b), and for symbol a, $left(a) = 1$ and $Right(a) = 14$.



(a) (b)

Figure 2.2: (a) Pseudoknotted secondary structure of length 13. (b) Arc diagram representation of the structure in part (a)

2.2 Pseudoknotted Structural Elements

Here we introduce two new notions in a secondary structure: *closed region* and *band.* Roughly speaking, a region C is a contiguous set of base indices, and if C is closed then there is no base index in C which is paired with a base index which is not in C. C is a pseudoknotted closed region if there is no arc diagram representation of it such that no two arcs cross each other. C is a pseudoknot free closed region if it is not a pseudoknotted closed region. Figure 2.1(b) and Figure 2.2(b) represent a pseudoknot free and a pseudoknotted closed region respectively. Within a pseudoknotted closed region, we define a notion of a *band.* Bands in a pseudoknotted closed region refer to pseudoknotted stems. In Figure 2.2(b), the base pairs (arcs) 1.11 and 2.10 form a band, and also the base pairs (arcs) 8.13 and 9.12 form a band. In the rest of the section we formally define the above notions.

All of the following definitions are with respect to a fixed secondary structure R with pattern representation P. We use $[i; j]$ to denote the set of indices $i, i + 1, ..., j$ and call it

a region if $i \leq j$. The union of two non-overlapping regions is called a *gapped* region.

Closed Regions: We say that $[i;j]$ $(i < j)$ is *weakly closed* if it contains at least one base pair and for all base pairs $i'.j'$ of R, $i' \in [i;j]$ if and only if $j' \in [i;j]$. We say that $[i;j]$ is *closed*, and write $i;j$, if either (i) $i = 1$ and $j = n$ or (ii) $[i;j]$ is weakly closed and for all l with $i < l < j$, $[i;l]$ and $[l;j]$ are not weakly closed. Each closed region corresponds to a substructure (subset of base pairs) of R, and also to a pattern. We say that a pattern is *closed* and refer to it as a *closed pattern* if its corresponding region is a *closed region.* For example, *abba* and *abab* are closed patterns, but *aabb* is not (unless *aabb* corresponds to the entire structure).

Let $[i;j']$ be a closed region. If i' and j are such that $i.j$ and $i'.j'$ then we say that $i.j$ and $i'.j'$ are the *external* base pairs of $[i;j']$. If $i.j'$ then the region has just one external base pair. We also refer to i and j' as $i;j'$'s left and right borders respectively.

Let $[i;j]$ and $[i';j']$ be closed with $i < i'$. If $j < i'$, we say that $[i;j]$ and $[i';j']$ are *disjoint*; otherwise we say that $[i';j']$ is *nested* in $[i;j]$.

For example, in Figure 2.3 $[12;36]$ is closed. $[15;34]$ is weakly closed but it is not closed as $[15;24]$ is weakly closed. Pattern $P = bcdeedcfggfb$ corresponds to $12;36$ and therefore is a closed pattern. 12.36 is the external base pair of $12;36$ where 12 and 36 are the left and the right borders of it respectively. $[15;24]$ and $[26;34]$ are disjoint closed regions and both are nested in $[12;36]$.

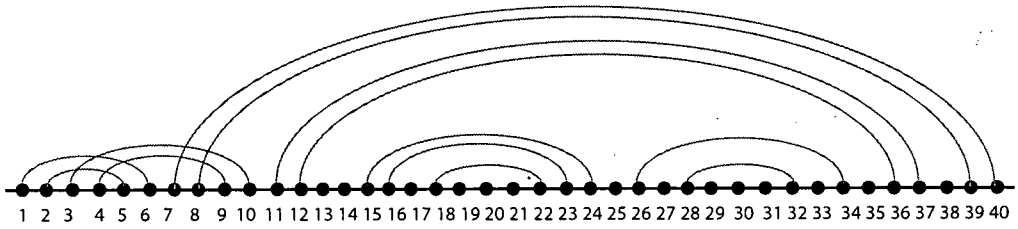
Pseudoknotted Closed Region: We say that $[i;j]$ is a pseudoknotted closed region of R if $i;j$ and $i.j \notin R$. We say that the indices i and j are the left and right borders of the pseudoknotted region $[i;j]$. Pair $i.j$ is pseudoknotted if there exists $i'.j'$ with $i < i' < j < j'$ or $i' < i < j' < j$. We also refer to i and j as *pseudoknotted* base indices. A pattern P is called a *pseudoknotted pattern* if it corresponds to a pseudoknotted closed region.

For example, in Figure 2.3 $[57;69]$ is a pseudoknotted closed region and its corresponding pattern $P = abccdebaed$ is a pseudoknotted pattern. 57.67 and 64.69 are its external base pairs, where 57 and 69 are the left and the right borders of it respectively. 57.67, 58.66, 64.69 and 65.68 are all pseudoknotted pairs and $57, 58, 64, 66, 67, 68$ and 69 are all pseudoknotted base indices.

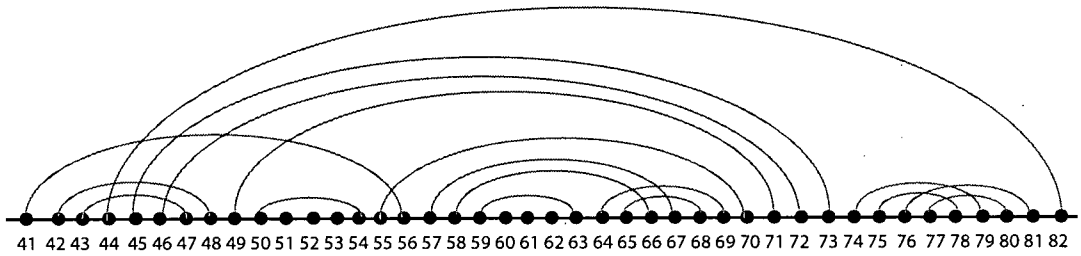
Pseudoknot Free Closed Region: We say that the closed region $i;j$ is pseudoknot free if there are no two pairs $i_1.j_1$ and $i_2.j_2$, with $i \leq i_1, j_1, i_2, j_2 \leq j$ such that either $i_1 < i_2 < j_1 < j_2$ or $i_2 < i_1 < j_2 < j_1$. A pattern P is called a *pseudoknot free pattern* if it corresponds to a pseudoknot free closed region.

For example in Figure 2.3 $[12;36]$ is a pseudoknot free closed region and its corresponding pattern $P = bcdeedcfggfb$ is a pseudoknot free pattern.

(a)



(b)



(c)

Figure 2.3: (a) Arc diagram representation of an RNA secondary structure R. R consists of two closed regions $[1;40]$ and $[41;82]$. (b)$[1;40]$ closed region. (c) $[41;82]$ closed region.

Bands: Let $i_2.j_2$ be a pseudoknotted base pair. We say that $i_2.j_2$ is *directly* banded in $i_1.j_1$ if (i) $i_1 \leq i_2 < j_2 \leq j_1$, and (ii) $[i_1+1, i_2-1]$ and $[j_2+1, j_1-1]$ are weakly closed. Note that the "is directly banded in" relation is reflextive. We let "are banded" be the symmetric and transitive closure of the "is directly banded in" relation. Let B be an equivalence class under the "are banded" relation. That is, B is a set of base pairs such that every two base pairs in B are banded and every base pair in B is pseudoknotted. B has closing base pairs $i_1.j_1$ and $i_2.j_2$ such that for every base pair $i.j$ in B, $i_1 \leq i \leq i_2$ and $j_2 \leq j \leq j_1$. Note that $i_1.j_1$ may equal $i_2.j_2$.

A gapped region $[i_1;i'_1] \cup [j'_1;j_1]$ is a *band* of a pseudoknotted region if for some equivalence class B, $i_1.j_1$ and $i'_1.j'_1$ are the closing pairs of B. Base pair $i.j$ or closed

region $i;j$ is contained in band $[i_1;i'_1] \cup [j'_1;j_1]$, if and only if either $i,j \in [i_1;i'_1]$ or $i,j \in [j'_1;j_1]$. Then we simply say that $i.j$ is in a band region. Base pair $i.j$ spans band $[i_1;i'_1] \cup [j'_1;j_1]$ if $i_1 \leq i \leq i'$ and $j'_1 \leq j \leq j_1$. We say that $[i_1;i'_1] \cup [j'_1;j_1]$ is a *band* of pseudoknotted region $[i;j]$ if $i \leq i_1 \leq j_1 \leq j$ and there is no closed region $[p;q]$ with $i < p \leq i_1 < j_1 \leq q < j$.

We say that $i_1.j_1$ and $i'_1.j'_1$ are band's closing pairs. $i_1.j_1$ is the outer and $i'_1.j'_1$ is the inner closing pair of the band. We say that $[i_1;i'_1]$ and $[j'_1;j_1]$ are band regions. i_1 and i'_1 are the borders of $[i_1;i'_1]$ band region and $j'_1 and j_1$ are the borders of $[j'_1;j_1]$ band region. We refer to i_1 and j_1 by the left and the right border of the band respectively.

For example in Figure 2.3 $[57;58] \cup [66;67]$ is a band of pseudoknotted closed region $[57;69]$ and 57.67 and 58.66 are the outer and the inner closing pair of the band. $[57;58]$ and $[66;67]$ are band regions where 57 and 67 are the left and the right border of the band. 57.67 and 58.66 both span the band.

We can infer the following corollaries from the above definitions:

Corollary 2.2.1 *A weakly closed region C can be decomposed to $C_1 \cup C_2 \cup ... \cup C_n \cup C'$ where each C_i ,$1 \leq i \leq n$, is a closed region and C' is a set of unpaired base indices.*

Corollary 2.2.2 *Assume that $i_1.j_1, i_2.j_2, ..., i_n.j_n$, $i_1 < i_2 < ... < i_n$, are the set of base pairs that spans a band, $[i_1;i'_1] \cup [j'_1;j_1]$. Then we have $j_n < .... < j_2 < j_1$.*

Corollary 2.2.3 *Each base pair $i.j$ in a pseudoknotted closed region either spans a band or belongs to some nested closed region.*

2.3 Loops in Pseudoknotted Structure

According to thermodynamic models of RNA secondary structure, the free energy of the secondary structure of an RNA is calculated based on the summation of free energy of all the loops in it (Figure 1.2).

Our definitions of hairpin and interior loops given below are standard for pseudoknot free structures. The definition of multi loops and external loop is generalized:

Hairpin loop: A *hairpin loop* contains a single external base pair $i.j$ and all bases in $[i+1, j-1]$ must be unpaired. For example, in Figure 2.3 18.22 is a hairpin loop external base pair.

Interior loop: An *interior loop* contains two unpseudoknotted base pairs $i.j$ and $i'.j'$ where $i < j' < j' < j$ and all bases in $[i+1, i'-1] \cup [j'+1, j-1]$ are unpaired. We refer to $i.j$ and $i'.j'$ as the interior loop external and internal base pairs respectively. There are two special cases of interior loops: *stacked pairs*, for which $i' = i+1$ and $j' = j-1$ and *bulge loops*, for which either $i' = i+1$ or $j' = j-1$ (but not both).

For example, in Figure 2.3 26.34 and 28.32 are the external and internal base pairs of an interior loop. 11.37 and 12.36 form a stacked pair, and 16.23 and 18.22 form a bulge loop.

Multiloop: A *multiloop* contains an external base pair $i.j$ and $k \geq 1$ tuples (i_1, j_1), (i_2, j_2), ..., (i_k, j_k) where $i_l; j_l, 1 \leq l \leq k$ and $i < i_1 < j_1 < i_2 < j_2 < ... < i_k < j_k < j$ and all bases in $[i, j] - \cup[i_l, j_l], 1 \leq l \leq k$ are unpaired. Furthermore, if $i_l.j_l$, $1 \leq l \leq k$ then k should be at least 2.

In Figure 2.3 12.36 is a multiloop external base pair with $(15, 24)$ and $(26, 34)$ as tuples.

External loop: An *external loop* contains $k > 0$ tuples (i_1, j_1), (i_2, j_2), ..., (i_k, j_k) where $i_l; j_l$, $1 < l < k$ and $i_1 < j_1 < i_2 < j_2 < ... < i_k < j_k$, along with the bases in $[1; n] - \cup_{1 \leq l \leq k}[i_l; j_l]$, all of which must be unpaired.

In Figure 2.3, $(1, 40)$ and $(41, 82)$ are the tuples of an external loop tuples.

We next introduce further types of elementary structures which are the consequence of having pseudoknotted base pairs and pseudoknotted regions. Several algorithms that predict pseudoknotted secondary structures, implicitly assign energies to these types of loops.

Pseudoknotted loop: Assume that $[i; j']$ is pseudoknotted region. Let $[i_1; i'_1] \cup [j'_1; j_1] \cup [i_2; i'_2] \cup [j'_2; j_2], ..., [i_m; i'_m] \cup [j'_m; j_m]$ be bands of $[i; j']$. Let $[p_1; q_1]$, $[p_2; q_2]$, ..., $[p_k; q_k]$ be closed regions which are nested in $[i; j] - (\cup_{l=1}^{m}[i_l; i'_l] \cup \cup_{l=1}^{m}[j'_l; j_l])$. The *pseudoknotted loop* corresponding to $[i; j']$ is the set $\{(i_l, j_l), (i'_l, j'_l) | 1 \leq l \leq m\} \cup \{(p_l, q_l) | 1 \leq l \leq k\}$, together with the bases in

$$[i, j'] - \cup_{l=1}^{k}[p_l; q_l] - \cup_{l=1}^{m}[i_l; i'_l] - \cup_{l=1}^{m}[j'_l; j_l]$$

all of which must be unpaired.

For example, in Figure 2.3, $[1; 40]$ corresponds to a pseudoknotted loop with $[1; 2] \cup [5; 6] \cup [3; 4] \cup [9; 10] \cup [7; 8] \cup [39; 40]$ as bands and $[11; 37]$ as the closed region nested in it; $[41; 82]$ also corresponds to a pseudoknotted loop with $[41; 41] \cup [56; 56] \cup [42; 43] \cup [47; 48] \cup [44; 46] \cup [72; 82] \cup [49; 55] \cup [70; 71]$ as bands and $[57; 69]$ as the closed region nested in it.

Interior-pseudoknotted loops: An *interior-pseudoknotted loop* contains two base pairs $i.j$ and $i'.j'$ where $i < i' < j' < j$, all bases in $[i+1, i'-1] \cup [j'+1, j-1]$ are unpaired and there is a band $[bi; bi'] \cup [bj'; bj]$ such that $bi \leq i < bi'$ and $bj' < j \leq bj$. We refer to $i.j$ and $i'.j'$ as the interior-pseudoknotted loop external and internal base pairs respectively.

For example, in Figure 2.3 1.6 and 2.5 are the external and internal base pairs of an interior-pseudoknotted loop.

Multi-pseudoknotted loops: A *multi-pseudoknotted loop* contains an external base pair $i.j$ and $k > 1$ tuples (i_1, j_1), (i_2, j_2), ..., (i_k, j_k) where: (i) $i_l; j_l$, $1 \leq l \leq k$ and $i < i_1 < j_1 < i_2 < j_2 < ... < i_k < j_k < j$, along with the bases in $[i; j] - \cup_{1 \leq l \leq k}[i_l; j_l]$, all of which must be unpaired, (ii) there is a band $[bi; bi'] \cup [bj'; bj]$ such that $bi \leq i < bi'$ and $bj' < j \leq bj$, and (iii) there is exactly one tuple (i_l, j_l) for which $i_l; j_l$ is not true (i.e $[i_l; j_l]$ is not a closed region) and $i_l.j_l$ spans a band ($bi \leq i_l \leq< bi'$ and $bj' \leq j_l \leq< bj$).

For example, in Figure 2.3 44.82 is the external base pair of a multi-loop which has $(45, 73)$ and $(74, 81)$ as tuples.

Multi-Pseudoknotted and Interior-pseudoknotted loops are called spanning band loops. If, in the above definition of interior loop, we remove the constraint that $i.j$ and $i'.j'$ should be unpseudoknotted, then interior-pseudoknotted loop would be an special case of interior loop. Also if for exactly one of the tuples (i_l, j_l) (where $i_1.j_1$) in the above definition of multiloop we allow $[i_l; j_l]$ not to be closed, then multi-pseudoknotted loop would be a special case of multiloop.

We can infer the following corollaries from the above definition and the definitions of Section 2.2:

Corollary 2.3.1 *Every unpaired base is in exactly one loop and every base pair is in exactly two loops. An external loop is a weakly closed region. Every other loop, except interior-pseudoknotted and multi-pseudoknotted loops, corresponds to a closed region. Interior-pseudoknotted and multi-pseudoknotted loops are not closed as their external base pair spans a band. Hairpin, interior and multi loops correspond to unpseudoknotted closed regions and pseudoknotted loops correspond to pseudoknotted closed regions.*

Corollary 2.3.2 *Every spanning band base pair, except the inner closing pair of the band, is either an interior-pseudoknotted or a multi-pseudoknotted loop external base pair.*

2.3.1 Location Attribute of Loops

By introducing bands, each loop nested in a pseudoknot loop gets a location status as follows which is needed in calculating its free energy.

in-Band loops: A Loop corresponding to a closed region which is nested in one of a bands' regions is an in-Band loop. The closing pairs of such a loop is in a band region.

In Figure 2.3 $50; 54$ corresponds to an in-Band hairpin loop and $[74; 81]$ corresponds to an in-Band pseudoknotted loop.

out-Band loops: Let PR denote a pseudoknotted region. A loop corresponding to a closed region which is nested in PR but is not nested in any band of PR is an out-Band loop.

In Figure 2.3 11; 37 corresponds to an out-Band multiloop and [57; 69] corresponds to an out-Band pseudoknotted loop.

span-Band loops: A Loop which is nested in a pseudoknotted region, but neither is in-Band nor out-Band is a span-Band loop. This kind of loop is either a Multi-pseudoknotted loop or an Interior-pseudoknotted loop.

For example, in Figure 2.3 44; 82 corresponds to a multi-pseudoknotted loop and 1; 6 corresponds to an interior-pseudoknotted loop.

2.4 Parse Tree

Let $[i; j]$ and $[i'; j']$ be closed regions. We say that $[i'; j']$ is a child of $[i; j]$ if $[i'; j']$ is nested in $[i; j]$ and is not nested in any $[i''; j'']$ with $i < i''$. We say that $[i; j]$ and $[i'; j']$ are siblings if they are children of the same closed region and $i \neq i'$. So the closed regions form a tree structure.

An ordered tree $T(R)$ is called the parse tree of R if: (i) there is a 1-1 correspondence between nodes of the tree and closed regions of R, and (ii) if node V corresponds to closed region C then V is the parent of all the nodes whose corresponding closed regions are nested in C. The children of each node are ordered by the left index of the closed region. We also refer to $T(R)$ by $T(P)$ where P is the pattern representation of R. Assume that C is the closed region corresponding to node V and $C_1, ..., C_m$ are the closed regions correspond to the children of V. Then we say that the pattern corresponding to C also corresponds to node V. Also, $C' = C - \cup_{i=1}^{m} C_i$ is called the *private region* corresponding to V and we refer to the pattern corresponding to C' as the *private pattern* of V. (Figure 2.4)

The parse tree of R contains information about R's closed regions, but it could be easily updated to contain other information such as band regions, loop types (see Chapter 5).

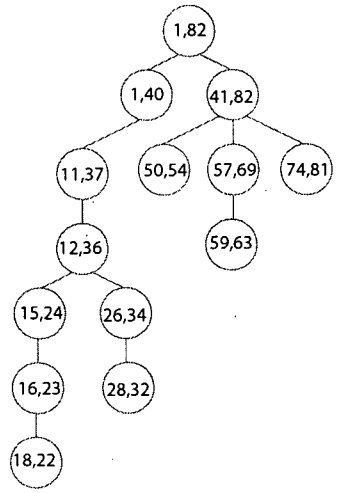


Figure 2.4: Parse tree for the structure in Figure 2.3. Consider node $V = (12, 36)$ as an example. $(15, 24)$ and $(26, 34)$ are V's children. $C = [12; 36]$ is V's corresponding closed region and its pattern is $P = bcdeedcfggfb$. C', the private closed region of V, contains 12.36 as base pair and its private closed pattern is $P' = bb$.

Chapter 3

Related Work

In this section, we describe related work in three cases. In Section 3.1 we describe work on classification of structural motifs that are found in pseudoknotted secondary structures. In Section 3.2 we discuss related work on algorithms for predicting pseudoknotted secondary structures, and means for classifying the structures handled by these algorithms. Related work on calculating the free energy of given secondary structure is discussed in Section 3.3. It should be noted that there is no related work on parsing a secondary structure, although parsing a pseudoknot free secondary structure is very similar to constructing an expression tree from an arithmetic expression whose parentheses are well-formed.

3.1 Classification Scheme

There are several works on pseudoknotted secondary structures which studied different possible structures that could be formed as a consequence of having pseudoknotted base pairs [12, 9]. They tried to classify pseudoknots (pseudoknotted substructures) into different types and study the types of pseudoknots which occur in known biological structures. However, their classification are incomplete and ambiguous (e.g. the definition of H-type and B-type pseudoknots) [10]. Our parsing algorithm can be used to classify different types of pseudoknots that arise in real structures. Although it is not part of this thesis, our notions of bands and our loop classifications can help to provide a precise definition for different type of pseudoknots.

3.2 Pseudoknotted Secondary Structure Prediction

General pseudoknotted RNA secondary structure prediction is NP-hard and therefore several incomplete polynomial complexity algorithms have been proposed to solve this problem. Dynamic programming and heuristic algorithms are two widely used approaches.

There are several dynamic programming algorithms for RNA secondary structure with pseudoknot prediction. Each of them can handle a restricted class of structures as

they try to find a minimum free energy (mfe) secondary structure for a limited classes of pseudoknotted structures.

Rivas and Eddy (R&E) [13] proposed an algorithm with a complexity of $O(n^6)$ in running time which can handle a large class of structures. They use different free energy parameters for the loops which are nested in a pseudoknot loop and consider a loop (interior or multi) $[i; j]$ to be nested in a pseudoknot if there is a band with two closing base pairs $i_l \cdot j_l$ and $i'_l \cdot j'_l$ such that $i_l \leq i \leq i'_l$ and $j'_l \leq j \leq j_l$ (span-band loop).

Uemura et al. (U) [15] proposed algorithms of time complexity $O(n^4)$ for simple pseudoknots and $O(n^5)$ or more for the other more complicated pseudoknots. Their algorithms are based on *tree adjoining grammar* (TAG) and therefore are not so easy to understand.

Akutsu (A) [2] re-formulated the method of Uemura et al. [15] and introduced simple dynamic programming algorithms for RNA secondary structure prediction. Akutsu's algorithms are of time complexity $O(n^4)$ for simple pseudoknots and $O(n^5)$ for more complicated pseudoknots. The algorithms are easy to understand and to modify and hence easier to cope with various score functions (i.e. free energy functions). However, the algorithms of Akutsu [2] handles a more restricted class of structures than does the algorithm of Uemura et al. (U) [15].

Dirks and Pierce (D&P) [7] described an $O(n^5)$ dynamic programming algorithm for computing the partition function and minimum free energy structure of an RNA (or DNA) strand. The class of pseudoknotted structures that their algorithm can handle consists of structures which pseudoknot loops have exactly two bands. They consider a loop to be nested in a pseudoknot loop (and hence use different parameters to calculate its free energy) if the loop is of the kind out-band.

Lyngso and Pederson (L&P) [11] introduced a simple $O(n^5)$ algorithm which can handle the structures with really simple pseudoknot loops. The class of structures they can handle allows pseudoknot loops with exactly two bands. They do not allow the pseudoknot loops nested in another one and also do not make a difference between the free energy of the loops whether they are nested in a pseudoknot loop or not.

It is proved by Condon et al.[5] that there is actually a relation between the class of structures each of these algorithms can handle : PKF $\subseteq$ L&P $\subset$ D&P $\subset$ A $\subset$ U $\subset$ R&E. (PKF represents the class of pseudoknot free structures)

This shows that there is actually a trade off between the complexity of the algorithm and the generality class of structures it can handle.

There are also several heuristic (and incomplete) algorithms for RNA secondary structure prediction [14, 8]. Heuristic algorithms find a locally minimum-energy secondary structure over the whole search space.

3.3 Free Energy Calculation

Heuristic algorithms commonly use a function for calculating the free energy of substructures (as partial solutions) in the process of searching for a low-energy structure in the search space. But there is no explicit work on calculating the free energy of given secondary structure in linear time. Here we can use our linear time free energy calculation algorithm, to calculate the free energy of partial solution substructures.

Chapter 4

Parsing Algorithm

In this chapter we introduce a linear time algorithm for parsing a secondary structure into its closed regions. A parsing algorithm creates the tree $T(R)$ of closed regions in R.

The algorithm for finding closed regions is described in Section 4.1. In Section 4.2, we prove that the closed region finding algorithm identifies all closed regions in the structure accurately. In Section 4.3 we explain how the parse tree is constructed gradually, by adding a node to the tree for each closed region, when it is identified. The last section explains the time complexity of the parsing algorithm, which is linear.

To build the tree of closed regions, our algorithm takes as input a *linked list representation* L of a secondary structure R for a strand of length n. In this representation, list elements are the base indices, with bidirectional links between adjacent elements and additionally bidirectional links between paired indices. For convenience, we define the function $bp(i)$ to be j if $i.j \in R$ or $j.i \in R$, and to 0 if i is unpaired.

4.1 Closed Region Finding Algorithm

The *Build-Closed-Regions-Tree* algorithm, presented in Algorithm 1, scans the list of indices from left to right and marks indices i with $i < bp(i)$, so as to identify closed regions efficiently. The tree of closed regions is built gradually by calling the *Add-To-Tree* procedure from the algorithm right after identifying each closed region.

We provide the intuition behind the algorithm in Section 4.1.1 and describe four invariants of it in section 4.1.2.

4.1.1 Intuition

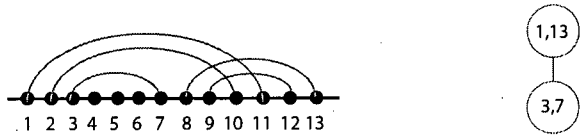
To identify a closed region it is enough to find its borders. Every unpseudoknotted base pair $,i.j$, is the external base pair of a closed region $,i;j$, and therefore i and j are the borders of $i;j$. But this is not true for pseudoknotted base pairs. According to the definition, a base pair $i.j$ is pseudoknotted if there exists $i'.j'$ with $i' < i < j' < j$ or $i < i' < j < j'$, where i in the former and i' in the latter case couldn't be a closed region border.

Scanning the list L of indices from left to right, if we reach $j = bp(i)$ knowing that there is an index i', $i < i' < j$, whose pair j' has not been scanned yet, we can conclude that both i and i' are pseudoknotted and i' is not a closed region border. To facilitate this procedure we introduce three types of marks for base indices, B, P and N. A base index $i < bp(i)$ is marked as (i) B (for "base pair found") if its base pair is found, (ii) P (for "pseudoknotted") if we know that it is pseudoknotted, and (iii) N ("not a border") if we know that it is not a border of a closed region.

For example, in Figure 4.1(a): (a) When we scan index 7, index 3 is B-marked as $7 = bp(3)$. (b) When we scan index 10, indices 8 and 9 are P-marked and N-marked as $2 < i < 10 < bp(i)$ for $i = 8$ and $i = 9$. Also, 2 is B-marked and P-marked as $10 = bp(2)$ and $2 < 8 < bp(2) < bp(8)$. Note that when index 10 has just been scanned, it is not clear what is the base pair of 1. Depending on the base pair of 1, 2 may or may not be the left border of a closed region, and so 2 is not N-marked yet. (c) When we scan index 11, 1 is B-marked and P-marked and 2 is N-marked, as now we know that 2 is in the same closed region as 1 and therefore 2 is not the left border of a closed region. More details on this example are given in Section 4.1.3.

To facilitate the process of marking, and keep it working efficiently, Algorithm 1 makes use of a stack ST. While scanning the list and marking base indices, for efficiency reasons the algorithm removes base indices from the list when they are no longer needed. The following base indices are removed from the list L: (i) base indices which are not the border of any closed region, and (ii) base indices which are the borders of an already identified closed region.

The stack and list also provide an efficient way to identify closed regions. For example, in Figure 4.1(a), when index 7 is scanned, its base pair, 3, is on top of the stack and this identifies that $[3; 7]$ is a closed region. Identifying pseudoknotted closed region is more complicated but is explained more fully below (Section 4.1.2 and Section 4.1.3).



(a) (b)

Figure 4.1: (a) Arc diagram representation of an RNA secondary structure. (b) Parse tree for the structure in part (a).

algorithm Build-Closed-Regions-Tree
input: linked list L representation of structure R for a strand of length n
output: tree T of closed regions of R

```
1  initialize T to contain one node labeled [1, n];
2  initialize stack ST to be empty;
3  λ := 1;
4  repeat
5    if λ < bp(λ) then
6      Push(λ, ST);

7    elseif bp(λ) = 0 then // λ is unpaired
8      Remove(λ, L);

9    else // bp(λ) < λ
10     B-mark(bp(λ));
11     if Top(ST) = bp(λ) =Pred(λ, L) then // (bp(λ), λ) is a closed region
12       Pop(ST);
13       if !(bp(λ) = 1 and λ = n) then Add-To-Tree([bp(λ), λ], T);
14       Remove(bp(λ), L);

15     elseif mark(Pred(bp(λ), L)) = P&B and bp(λ) = Pred(λ, L) then
         // λ is the right border of a pseudoknotted region
16       h ← Pop(ST); // note that h =Pred(bp(λ), L)
17       Remove(h, L);
18       if !(h = 1 and λ = n) then Add-To-Tree([h, λ], T);
19       Remove(bp(λ), L);

20     else
         // bp(λ) · λ is pseudoknotted but λ isn't the right border of a closed region
21       P-mark(bp(λ));
22       while Top(ST) > bp(λ) do
23         j ← Pop(ST);
24         P-mark(j);
25         N-mark(j);
26         if mark(j) = B&N&P then
27           Remove(j, L);
28       if mark(bp(λ)) = B&N&P then
29         Remove(bp(λ), L);
30     Remove(λ, L);
31   λ := λ + 1;
32 until λ = n + 1;
33 return T
```

Algorithm 1: Build the tree of closed regions.

4.1.2 Invariants

The following invariants are maintained when index l has just been scanned. The *Mark Invariant* formalizes what marks an index has when index l has just been scanned by the algorithm. The *List* and *Stack Invariants* formalize what is in the list L and stack ST when index l has just been scanned. Finally, the *Closed Region Invariant* formalizes how a closed region can be identified by the algorithm.

Mark Invariant: For all indices i in L with $i \leq l$ and $i < bp(i)$,

> **B-mark invariant:** i is B-marked if and only if $bp(i) \leq l$;
>
> **P-mark invariant:** i is P-marked if and only if for some i',
> $i < i' < bp(i) \leq l$ and $bp(i) < bp(i')$, or
> $i' < i < bp(i') \leq l$ and $bp(i') < bp(i)$;
>
> and
>
> **N-mark invariant:** i is N-marked if and only if there exists i' in the same closed region as i with $i' < i < bp(i') \leq l$.

List Invariant: For all $a \leq l$, a is not in list L if and only if either (i) a is B-marked, P-marked, and N-marked, (ii) $bp(a) < a$, or (iii) for some $b \leq l$, $a; b$.

Stack Invariant: ST contains exactly those indices $i \leq l$ such that i is in L and i is not N-marked.

Closed Region Invariant: If $h \neq 1$ or $l+1 \neq n$, then $[h, l+1]$ is a closed region of R if and only if $bp(l+1) < l+1$, h is on the top of the stack ST, $bp(l+1)$ is $l+1$'s predecessor in list L, and either (i) $h = bp(l+1)$ or (ii) h is B-marked and P-marked and h is $bp(l+1)$'s predecessor in list L.

Note that $[1; n]$ is a special case of closed region which is added to the $T(R)$ in the first step by the algorithm, so we excluded it from the closed region invariant and the following sections in this chapter.

4.1.3 Example

Consider the structure in Figure 4.1(a) as an example. Table 4.1 shows the list, stack, marks and closed region identified, as Algorithm 1 is executed on the example. (Here, we do not show the links in L, but they can be easily infered from the figure.)

When index 6 is scanned by the algorithm, it is removed from L as it is unpaired. Then $Top(ST) = 3 = bp(7) = Pred(7)$, and therefore algorithm identifies $(3, 7)$ as a closed region, and add $(3, 7)$ to the parse tree, when it scans 7.

When index $10 = bp(2)$ is scanned by the algorithm, indices 8 and 9 are P-marked and N-marked, as $2 < i < 10 < bp(i)$ for $i \in \{8, 9\}$, and are removed from the stack ST. Moreover, index 2 is B-marked and P-marked.

When the algorithm scans index $11 = bp(1)$, index 2 is N-marked and as it was P-marked and B-marked before it is removed from L (and therefore ST). Moreover, index 1 is B-marked and P-marked.

When index $12 = bp(9)$ is scanned by the algorithm, index 9 is B-marked and as it was P-marked and N-marked before it is removed from L. Then $8 = bp(13) = Pred(13)$ and $Top(ST) = 1 = pred(8)$ and 1 is P-marked and B-marked, but $13 = n$, therefore the algorithm does not identify $(1, 13)$ as a closed region, since $(1, 13)$ is already in the tree.

At the end, the parse tree for the structure in Figure 4.1(a) has $(1, 13)$ in the root and $(3, 7)$ as its child (Figure 4.1(b)).

λ	List L with marks on elements	Stack ST	Closed region identified
0	{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13}	{ }	
1	{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13}	{1}	
2	{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13}	{1, 2}	
3	{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13}	{1, 2,3}	
4	{1, 2, 3, , 5, 6, 7, 8, 9, 10, 11, 12, 13}	{1, 2, 3}	
5	{1, 2, 3, , , 6, 7, 8, 9, 10, 11, 12, 13}	{1, 2, 3}	
6	{1, 2, 3, , , , 7, 8, 9, 10, 11, 12, 13}	{1, 2, 3}	
7	{1, 2, , , , , , 8, 9, 10, 11, 12, 13}	{1, 2}	(3,7)
8	{1, 2, , , , , , 8, 9, 10, 11, 12, 13}	{1, 2, 8}	(3,7)
9	{1, 2, , , , , , 8, 9, 10, 11, 12, 13}	{1, 2, 8, 9}	(3,7)
10	{1, 2, , , , , , 8, 9, , 11, 12, 13} B P P P N N	{1, 2}	(3,7)
11	{1, , , , , , , 8, 9, , , 12, 13} B P P P N N	{1}	(3,7)
12	{1, , , , , , , 8, , , , , 13} B P P N	{1}	(3,7)
13	{ , , , , , , , , , , , , , }	{ }	(3,7)

Table 4.1: Execution of Algorithm 1 on secondary structure of Figure 4.1(a). There is one row for each base index, plus an initial row, 0. Column 1 shows which base index is scanned by the algorithm. The remaining columns describe what happens when index i is scanned (i.e when $\lambda = i$). Column 2 gives the list L along with the marks on list elements and column 3 gives the stack, when i has just been scanned (when the algorithm reaches line 31). Column 4 lists which closed region has been identified (and added to the tree) by the algorithm, if any, when i is scanned.

4.2 Closed Region Finding Algorithm: Proof

We prove in Theorem 4.2.1 that Algorithm 1 calls $Add - to - tree([i,j],R)$ if and only if $[i;j]$ is closed. First , we prove that the *closed region invariant*, and therefore the three other invariants, are maintained by the algorithm.

We prove in Claim 4.2.1 that all the above invariants are maintained by the algorithm, on an input list L that represents structure R for a strand of length n, when l has just been scanned. Formally, we say that l has just been scanned when line 32 of the algorithm is reached with $\lambda = l+1$ or line 4 is reached and $l = 0$. We also say l is scanned when lines 5 to 30 of the algorithm are executed with $\lambda = l$.

Claim 4.2.1 *For any input L all invariants are true when l has just been scanned, for all l, $0 \leq l \leq n$.*

Proof

We use induction to prove Claim 4.2.1. It is obviously true when we have $l = 0$ (base case).

Let $l > 0$. Assume that the claim is true for all $k < l$ (Induction Hypothesis). We prove that the claim is true for l.

4.2.1 Mark Invariant Proof

Here we prove that all three mark invariants are true.

B-mark: When l is scanned, if $i < bp(i) = l$ then i is B-marked on line 10 of the algorithm and no other index is B-marked. Since marks are never removed from an index, by the Induction Hypothesis the *B-mark invariant* is true for all i with $i < bp(i) < l$, therefore the *B-mark invariant* is true.

P-mark: We can decompose each case of the *P-mark invariant* into two subcases:

- The case "For some i', $i < i' < bp(i) \leq l$ and $bp(i) < bp(i')$" is decomposed to

  Case 1.1 For some i', $i < i' < bp(i) < l$ and $bp(i) < bp(i')$

  Case 1.2 For some i', $i < i' < bp(i) = l$ and $bp(i) < bp(i')$, which is equivalent to $bp(l) < i' < l < bp(i')$, where $bp(i) = l$.

- The case "For some i', $i' < i < bp(i') \leq l$ and $bp(i') < bp(i)$" is decomposed to

  Case 2.1 For some i', $i' < i < bp(i') < l$ and $bp(i') < bp(i)$

  Case 2.2 For some i', $i' < i < bp(i') = l$ and $bp(i') < bp(i)$ which is equivalent to $bp(l) < i < l < bp(i)$, where $bp(i') = l$.

Now we need to prove that i is P-marked if and only if one of the above four cases holds.

Part 1 i is P-marked if one the above four cases holds.

Proof

If one of the two first subcases, Case 1.1 or Case 2.1, holds then i is P-marked by the Induction Hypothesis.

Assume that the Case 1.2 holds, which means that $bp(l) < i' < l < bp(i')$. If $bp(l)$ is not P-marked when $l-1$ has just been scanned then one of the following occurs: (i) i' is not N-marked and therefore (by the *Stack invariant* and Induction Hypothesis) $bp(l)$ is not on the top of the stack. In this case the algorithm skips line 11. (ii) i' is N-marked. In this case there exists i'', in the same closed region as i', with $i'' < i' < bp(i'') < l$ (by the *N-mark invariant* and Induction Hypothesis). If $bp(l)$ is not already P-marked then (by the *P-mark invariant* and Induction Hypothesis) there is no i'' with $i'' < bp(l) < bp(i'') < l$. Therefore $bp(l) < i'' < l$. We choose the first such i''. This i'' can not be N-marked, otherwise it won't be the first such i'' (by the *N-mark invariant*). Therefore i'' is in L and separates $bp(l)$ from l in L. So $bp(l)$ is not l's predecessor and the algorithm skips line 15. As a result, the algorithm reaches line 20 and $bp(l)$ is P-marked in line 21.

As the last case, assume that Case 2.2 holds, which means that $bp(l) < i < l < bp(i)$. If i is not P-marked when $l-1$ has just been scanned then, using the same explanation as above, the algorithm skips lines 11 and 15 and reaches line 20. $bp(l) < i < l < bp(i)$ and i is not P-marked, so i is not N-marked (otherwise it is N-marked in line 25 and therefore P-marked in line 24). Therefore i is still on the stack (by the *List, Stack* and *Mark invariants*) and it is above $bp(l)$ on the stack. So i is P-marked in line 24.

□

Part 2 One of the above four cases holds if i is P-marked.

Proof

If i is P-marked when $l-1$ has just been scanned then by the Induction hypothesis one of the two first subcases, Case 1.1 or Case 2.1, holds.

Assume that i is not P-marked when $l-1$ has just been scanned and it receives a P-mark when l is scanned. i can receive a P-mark only in lines 21 or 24.

Assume that i receives P-mark in line 21, which means that $i = bp(l)$. As the algorithm reaches that line, which means that it skipped lines 11 and 15, so $[bp(l), l]$ is not a closed region and there exists an i' such that either $i' < bp(l) < bp(i') < l$ or $bp(l) < i' < l < bp(i')$ (by the closed region definition). The former case means that $bp(l)$ is already P-marked , contradicting the fact that $bp(l)$ is not P-marked yet, therefore the latter case is true and Case 1.2 holds.

As in the last case, assume that i receives P-mark in line 24, where $i \neq bp(l)$. It means that i is above $bp(l)$ on the stack and therefore $bp(l) < i < l$. Also

$l < bp(i)$, otherwise either $[i, bp(i)]$ is a closed region, and therefore i is not on the stack (by the *list* and the *Stack invariants*), or $[i, bp(i)]$ is not a closed region and (by the closed region definition) $(i, bp(i))$ is a pseudoknotted base pair and i is already P-marked. Therefore $bp(l) < i < l < bp(i)$ and Case 2.2 holds. □

N-mark: We need to prove that i is N-marked if and only if there exists i' in the same closed region as i with $i' < i < bp(i') \leq l$.

Part 1 i is N-marked if there exists i' in the same closed region as i with $i' < i < bp(i') \leq l$.

Proof

If there exists i' in the same closed region as i with $i' < i < bp(i') < l$ then by the Induction Hypothesis i is N-marked when $l-1$ has just been scanned.

Assume that i is not N-marked when $l-1$ has just been scanned and $i' < i < bp(i') = l$. If $bp(l) < i < l < bp(i)$ then using the same proof as *Part 1 - Case 2.2* in above (by the *P-mark invariant*) i is N-marked in line 25. If $bp(l) < i < bp(i) < l$ then i is still on the stack (by the *List* and the *Stack invariants*) and in the list L. So $bp(l)$ is not on the top of the stack and the algorithm skips line 11. Furthermore i separates $bp(l)$ and l in L so $bp(l)$ is not the predecessor of l which causes the algorithm to skip line 15. As a result, the algorithm runs line 20 and i is N-marked in line 25.

□

Part 2 There exists i' in the same closed region as i with $i' < i < bp(i') \leq l$ if i is N-marked.

Proof

If i is N-marked when $l-1$ has just been scanned then by the Induction Hypothesis there exists i' in the same closed region as i, with $i' < i < bp(i') < l$.

Assume that i is not N-marked when $l-1$ has just been scanned and it receives an N-mark when l is scanned. i can receive an N-mark only in line 25. In this case i should be on the stack above $bp(l)$ which means that $bp(l) < i < l$. If $l < bp(i)$ then it is clear that i is in the same closed region as $bp(l)$. Otherwise, if $bp(i) < l$, as i is still on the stack it should be in the same closed region as $bp(l)$ (by the *List* and the *Stack invariants*). Therefore $bp(l) < i < l$ and i is in the same closed region as $bp(l)$.

□

Having all of these, the *Mark invariant* is true.

4.2.2 List Invariant Proof

We need to prove that for all $a \leq l$, a is not in the list L if and only if (i) a is B-marked, P-marked, and N-marked, (ii) $bp(a) < a$, or (iii) for some $b \leq l$, $a; b$.

Part 1 a is not in L if any of the above three cases holds.

Proof

If $a < l$ then by Induction Hypothesis when $l - 1$ has just been scanned a is not in L if either (i) a is B-marked, P-marked and N-marked, (ii) $bp(a) < a$, or (iii) for some $b < l$, $a; b$.

Now we need to prove that by scanning l, (i) $a \leq l$ is removed from L if a receives its last mark, which could be either N, P or B, (ii) l is removed from L if $bp(l) < l$, and also (iii) index h, $h < l$, is removed from L if $h; l$.

(i) Assume that when l is scanned a receives its last mark, which is either N, P or B. If N is the last mark then a should get it in line 25 and the algorithm checks right after that, in line 26, whether a has all three marks and removes it. If P is the last mark, l should get it either in line 21 or 24. In the former case the algorithm checks in line 28 and in the latter checks in line 26 whether a has all three marks and removes it. If B is the last mark then a should get it in line 10 and $a = bp(l)$. Then the algorithm continues either from line 15 or 20 (the algorithm should skip line 11 as $a = bp(l)$ is N-marked before and by the *Stack invariant* $a = bp(l)$ is not on the stack). If the algorithm runs line 15 then $a = bp(l)$ is removed in line 19, and if it runs line 20 then $a = bp(l)$ is removed in line 29.

(ii) Assume that $bp(l) < l$. Then l is removed from the list either by line 8, if l is not paired, or line 30, if l is paired.

(iii) Assume that there exists $h < l$ such that $h; l$. Then l is removed from the list either in line 14 or line 17 by the *Closed region invariant.*

□

Part 2 Either of the above three cases holds if a is not in L.

Proof

If $a < l$ is not in the list L when $l - 1$ has just been scanned then either (i) a is B-marked, P-marked and N-marked, (ii) $bp(a) < a$, or (iii) for some $b < l$, $a; b$.

Assume that a is removed from L when l is scanned. An index is removed from L in either of these lines: $8, 14, 17, 19, 27, 29$ or 30.

(i) If a is removed from L either in line 19, line 27 or line 29 then a has all three marks, P, N and B.

(ii) If a is removed from L either in line 8 or line 30 then $a = l$ and $bp(a) < a$.

(iii) If a is removed from L either in line 14 or line 17 then by the *Closed region invariant* there exists $h < l$ such that $h; l$.

□

4.2.3 Stack Invariant Proof

When the algorithm removes an index i from the list L it removes i from the stack too (lines 12 and 16). The algorithm also removes all indices, which are N-marked, from the stack in line 23. Furthermore, an index is not removed from the stack in any other place in the algorithm. So the *stack invariant* is maintained.

4.2.4 Closed Region Invariant Proof

We need to prove that $[h, l+1]$ is a closed region of R if and only if $bp(l+1) < l+1$, h is on the top of the stack ST, $bp(l+1)$ is $l+1$'s predecessor, and either (i) $h = bp(l+1)$ or (ii) h is B-marked and P-marked and h is $bp(l+1)$'s predecessor.

Part 1 $[h, l+1]$ is a closed region if one of the above two cases holds.

Proof

(i) Assume that when l has been just scanned $h = bp(l+1) < l+1$ is on the top of the stack and is $l+1$'s predecessor. It means that there is no i' such that $h < i' < l+1 < bp(i')$ (otherwise by the *List invariant*, i' is still in the list and h could not be $l+1$'s predecessor) or $i' < h < bp(i') < l+1$ (otherwise by the *N-mark invariant* h is N-marked and therefore by the *Stack invariant* should not be on the stack). Therefore $[h, l+1]$ is a closed region.

(ii) Assume that h is on the top of the stack and is P-marked and B-marked. Therefore there is no i' such that $i' < h < bp(i') < l$ and h is in the same closed region as i'. (Otherwise h is N-marked, by the *N-mark invariant*). Suppose that h is $bp(l+1)$'s predecessor which is $l+1$'s predecessor. It means that there is no base pair i' such that $h < i' < l+1 < bp(i')$ (otherwise i' is on the top of the stack by the *Stack invariant*). Therefore $[h, l+1]$ is a closed region.

□

Part 2 One of the above two cases holds if $[h, l+1]$ is a closed region.

Proof

$[h, l+1]$ is a closed region so $bp(l+1) < l+1$. h is either paired with $l+1$ or not :

(i) Assume that $h = bp(l+1)$. Then, by Lemma 4.2.1 below, there is no index remaining in L between h and $l+1$. So h is on the top of the stack (by the *Stack invariant*) and it is $l+1$'s predecessor. Therefore the first case, (i), of the *Closed region invariant* is satisfied.

(ii) Assume that h is not paired with $l+1$. Then $[h, l+1]$ is a pseudoknotted closed region. It is clear, by Lemma 4.2.1, that the only index remaining in L between h and $l+1$ is $bp(l+1)$. $[bp(l+1); l+1]$ is not a closed region (otherwise $[h, l+1]$ is not a closed region by the closed region definition) and as $[h, l+1]$ is a closed region so there is no index i' such that $bp(l+1) < i' < l+1 < bp(i')$. Therefore, there is an index i' such that $i' < bp(l+1) < bp(i') < l+1$ which means that $bp(l+1)$ is N-marked (by the *N-mark invariant*) and is not on the stack. So h is on the top of the stack (by the *Stack invariant*) , h is P-marked and B-marked (by the *Mark invariant* and closed region definition) and it is $bp(l+1)$'s predecessor in the list (by the *List invariant*), which is $l+1$'s predecessor itself. Therefore the second case, (ii), of the *Closed region invariant* is satisfied.

□

Lemma 4.2.1 *If $[h; l]$ is a closed region then, when $l-1$ has just been scanned, the only indices a, $h \leq a \leq l$, which remain in L are : h, l and $bp(l)$.*

Proof

All a such that $h < bp(a) < a < l$ are removed from L when $l-1$ has just been scanned by the *List invariant*. All a such that $h < a < bp(a) < l$ are B-marked (by the *B-mark invariant*). If a is just B-marked then there is no i' such that $i' < a < bp(i') < bp(a)$ or $a < i' < bp(a) < bp(i')$ (by the *P-mark* and *N-mark invariants*) and therefore $[a, bp(a)]$ is a closed region and a is removed from L by *List invariant* . If a is P-marked too, then either $a; b$ (for some b, $b < l$) or a is N-marked too (by the *N-mark invariant*). In both cases a is removed from L by the *List invariant*. So the only indices which remain are: h, l and $bp(l)$.

□

□

4.2.5 Add-to-tree Calls Theorem

Theorem 4.2.1 *On input R, Algorithm 1 calls* Add-to-tree$([h, l+1], R)$ *if and only if $[h, l+1]$ is a closed region of R.*

Proof By the *closed region invariant*, the proof is as follows: We will show that Algorithm1 calls *Add-to-tree*$([h, l+1], R)$ if and only if when l has just been scanned,

$bp(l+1) < l+1$, h is on the top of the stack ST, $bp(l+1)$ is $l+1$'s predecessor, and either (i) $h = bp(l+1)$ or (ii) h is B-marked and P-marked and h is $bp(l+1)$'s predecessor.

Part 1 Algorithm1 calls *Add-to-tree*$([h, l+1], R)$ if one of the above two cases holds.

Proof

Assume that the first case holds, so when l has just been scanned, $bp(l+1) < l+1$, h is on the top of the stack ST, $h = bp(l+1)$ and h is $l+1$'s predecessor. So while scanning $l+1$, the algorithm runs line 10 and then as all conditions in line 11 are satisfied it calls *Add-to-tree*$([h, l+1], R)$.

Now assume that the second case holds, so when l has just been scanned, $bp(l+1) < l+1$, h is on the top of the stack ST and it is B-marked and P-marked and also is $bp(l+1)$'s predecessor, where $bp(l+1)$ is $l+1$'s predecessor. So scanning $l+1$ algorithm runs line 10 and then skips line 11 as h is on the top of the stack and not $bp(l+1)$. Then as all conditions in line 15 are satisfied the algorithm calls *Add-to-tree*$([h, l+1], R)$.

□

Part 2 One of the above two cases holds if Algorithm1 calls *Add-to-tree*$([h, l+1], R)$.

Proof

The algorithm calls *Add-to-tree*$([h, l+1], R)$ if it reaches either line 13 or 18 when scanning $l+1$.

The algorithm reaches line 13 only if $bp(l+1)$ is on the top of the stack and $bp(l+1)$ =Pred$((l+1), L)$. In this case it identifies $[bp(l+1), l+1]$ as a closed region which means that $h = bp(l+1)$. So the first case holds.

The algorithm reaches line 18 (and skips line 11) only if $bp(l+1)$ is not on the top of the stack, $bp(l+1)$ =Pred$(l+1)$ and Pred$(bp(l+1), L)$ is marked as $P\&B$. In this case it identifies $[h, l+1]$ as a closed region where h is on the top of the stack. Note that h should be the predecessor of $(bp(l+1), L)$. So the second case holds.

□

□

4.3 Constructing the Parse Tree

The Parse tree of structure R is built gradually by calling *Add-To-Tree* procedure from *Closed region finding* algorithm whenever a closed region is identified.

The *Add-To-Tree*$([h, l], T)$ procedure works as follows:

- Create a new node $[h, l]$ if $[h, l] \neq [1, n]$.

- For each child $[a, b]$ of the root $[1, n]$ of T, check whether $a > h$, and make $[a, b]$ a child of $[h, l]$ (and thus no longer a child of $[1, n]$) if this is the case.
- Make $[h, l]$ a child of $[1, n]$.

By maintaining a list of the children of each node ordered by the left border index, and traversing them from the greatest to the smallest, this can be done efficiently in linear time in the number of children $[h, l]$ has.

4.4 Parsing Algorithm Running Time

Algorithm 1 scans the whole list once from left to right. So each index is visited once, and is pushed on the stack, popped off the stack, removed from the list, and receives each mark at most once. Therefore Algorithm 1 accesses each index for a constant number of times, and ignoring the cost of calls to the *Add-To-Tree* procedure, it is linear in the number of indices, n, in the list.

The *Add-To-Tree*$([h, l], T)$ procedure is linear in the number of children $[h, l]$ has. So the whole cost of calls to *Add-To-Tree* procedure is linear in the number of nodes in the tree T, which is at most n.

Therefore the parsing tree algorithm is *linear* in the number of indices, n, in the list and thus in the number of base pairs in the structure R.

Chapter 5

Energy Calculation

Standard thermodynamic models calculate the free energy of each loop using certain information about the loop, such as: (i) its external base pair, (ii) an ordered list of its base pairs or tuples, (iii) number of unpaired bases in the loop, and (iv) type and the location of the loop (see Section 2.3.1). The free energy of a secondary structure is calculated by summing the free energy of its component loops.

Here we represent an algorithm that identifies all loops in the structure plus needed information about them in linear time, using the parsing algorithm. Then all we need to calculate the the free energy of the structure is to add up the free energy of all loops, and clearly it can be done in linear time.

Given a structure R of length n, the parsing algorithm creates the parse tree, $T(R)$, where we have $[1;n]$ in the root and every other node in the tree corresponds to a closed region of R. The external loop of R corresponds to the root node. Every other loop, except interior-pseudoknotted and multi-pseudoknotted loops, corresponds to a closed region and therefore to a node in the tree. We can easily derive all the information, except the location, for unpseudoknotted loops from $T(R)$ in linear time.

Interior-pseudoknotted and multi-pseudoknotted loops are not closed as their external base pair spans the band. To identify these types of loops, the location status of all the other loops, and also to complete the information needed for pseudoknotted loops, we need to find the band regions for each pseudoknotted closed region(pseudoknotted loops).

Having band regions for a pseudoknotted closed region $i;j$ and knowing the location status of its children, we have all needed information for $i;j$'s corresponding pseudoknotted loop.

The algorithm does the following steps to achieve the above goals.

Loop type finding: The algorithm decides the type of loop corresponding to each closed region.

Band finding: The algorithm finds the list of bands for each pseudoknotted closed region. At the end of this step an ordered list of bands regions (ordered by the left border

index) is assigned to the corresponding pseudoknotted closed region.

Loop location finding: The algorithm figures out the location status of all loops in the tree, using the ordered list of band regions.

Inner loop finding: The algorithm identifies all Multi-pseudoknotted and Interior-pseudoknotted loops, which we refer to by *Inner Loops*, in addition to their necessary information.

5.1 Loop Type Finding

Each closed region $i;j$ corresponds to a loop whose type can be determined as follows, using the definition of each loop (Chapter 2).

- **Pseudoknot loop** Closed region $i;j$ corresponds to a pseudoknotted loop if $i.j \notin R$ ($i;j$ is a pseudoknotted closed region).
- **Hairpin Loop** Pseudoknot free closed region $i;j$ corresponds to a hairpin loop if it has no children in $T(R)$.
- **Interior Loop** Pseudoknot free closed region $i;j$ corresponds to an interior loop if it has exactly one child in $T(R)$, which is also not a pseudoknotted closed region.
- **Multi Loop** Pseudoknot free closed region $i;j$ corresponds to a multiloop if either (i) it has more than one child in $T(R)$, or (ii) it has one child which is a pseudoknotted closed region.

5.2 Band Finding

Our *band finding* algorithm is given in Algorithm 2.

Let $C = i;j$ be a pseudoknotted closed region in R. We defined a *linked list representation* L for R, in Chapter 4. Now let BL be a sublist of L starting from index i to index j. In the first step we remove the following from BL: (i) unpaired base indices, and (ii) base indices corresponding to nested closed regions.

Then by Corollary 2.2.3, BL only contains indices corresponding to spanning band base pairs. In other words, BL is a *linked list representation* of spanning band base pairs in $i;j$.

Inspired by Corollary 2.2.2, Algorithm 2 scans the list BL from left to right to identify bands and their region's borders.

Next-leftBase(b'_i, BL) returns i, the first index after b'_i in BL for which $bp(i) > i$. $i.bp(i)$ will be the outer closing pair of the next band.

We could easily get the ordered list of band regions, *BandRegions* (ordered by the left border index) by doing the following after identifying each band: (i) removing the base

algorithm Band-Finding
input: linked list BL representation of spanning band base pairs in $i; j$
1 $b_i := i$;
2 **repeat**
3 $b_j := bp(b_i)$; // $b_i.b_j$ is the outer closing pair of a band, B
4 $b'_i := b_i$;
5 $b'_j := b'_j$;
7 **while** Next$(b'_i, BL) = bp($Prev$(b'_j, BL))$ **do**
8 $b'_i :=$ Next(b'_i, BL);
9 $b'_j :=$ Prev(b'_j, BL);
// $b'_i.b'_j$ is the inner closing pair of the band B
// So $B = [b_i; b'_i] \cup [b'_j; b_j]$ is a band of $i; j$
10 $b_i :=$Next-leftBase(b'_i, BL);
11 until $b_i = j + 1$;

Algorithm 2: Find Bands of a pseudoknotted closed region.

indices corresponding to the band regions from BL, and instead (ii) storing the band regions borders.

For example, in Figure 5.1 BL corresponding to $1; 42$ includes the following base pairs: 1.16, 2.8, 3.7, 4.42, 5.33, 6.32, 9.31 and 15.30. Algorithm 2 figures out the bands in the following order: (1) $[1; 1] \cup [16; 16]$, (2) $[2; 3] \cup [7; 8]$, (3) $[4; 6] \cup [32; 42]$, and $[9; 15] \cup [30; 31]$. So we have the ordered list of band regions as
$BandRegions = \{[1; 1], [2; 3], [4; 6], [7; 8], [9; 15], [16; 16], [30; 31], [32; 42]\}$.

Running Time:
We can do the first step for all of the closed regions by first removing all unpaired bases from L and then removing nested closed regions (which all have already been identified) from each closed region. A closed region $i'; j'$ can be removed (from its parent closed region) in constant time by making Prev(i', L) and Next(j', L) to be adjacent (by updating the links). The former is linear in n, number of base pairs, and the latter is linear in the number of nodes in the tree, which is at most n. So the whole cost of the first is linear in n (number of base pairs in R).

Algorithm 2 scans BL,the spanning band base pairs list, once. So it is linear in the size of BL. Therefore it takes $O(n)$ time for all of the closed regions.

Thus, finding *BandRegions* list for all pseudoknotted closed regions in R is linear in n (number of base pairs in R).

5.3 Loop Location Finding

For pseudoknotted closed region C, let *ChildList* be the list of its children ordered by the left border index. So having *BandRegions*, an ordered list of band regions, (Section 5.2)

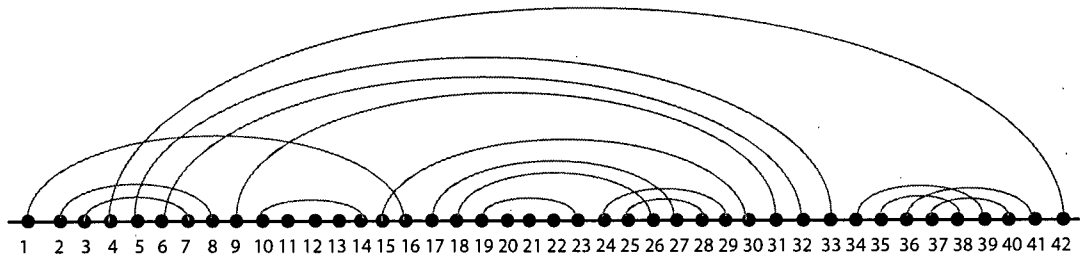


Figure 5.1: Arc diagram representation of an RNA secondary structure (closed region [41; 82] in Figure 2.3(c)).

we can easily figure out the location status of C's children. All we need to do is to scan both ordered list from left to right (in parallel, similar to Merge Sort procedure) and check whether a child is nested in a band region or not. A child corresponds to a *in-Band* loop in the former and to a *out-Band* loop in the latter case. This can be done in linear time in the size of *BandRegions* and *ChildList*. So the whole cost for all of the closed regions is linear in n (number of base pairs in R) plus the number of nodes in the tree, which is at most n. Therefore, *loop location finding* algorithm is linear in n (number of base pairs in R).

5.4 Inner Loop Finding

By Corollary 2.3.2, each spanning band base pair, except the inner closing pair of the bands, corresponds to either an interior-pseudoknotted or a multi-pseudoknotted loop external base pair.

Inspired by Corollary 2.3.2 and definitions for interior-pseudoknotted and multi-pseudoknotted loops (Chapter 2), the *Inner Loop Finding* algorithm does the following for each pseudoknotted closed region $i; j$ to identify all loops of interior-pseudoknotted and multi-pseudoknotted types:

1. Take the *link list representation* of spanning band base pairs, BL (Section 5.2), and ordered list of children, *ChildList* (Section 5.3) as input..

2. Similar to *loop location finding* algorithm (Section 5.3) scan BL and *ChildList* from left to right and assign to each index i the number of children which are nested between i and $Next(i)$, $Nested(i)$, along with the list of their indices, $Tuples(i)$.

3. Each index i, $bp(i) > i$, in BL corresponds to an interior-pseudoknotted loop with $i.bp(i)$ as the external and $\text{Next}(i, BL).\text{Prev}(bp(i), BL))$ as the internal base pairs, if the followings are satisfied: (i) $\text{Next}(i, BL) = bp(\text{Prev}(bp(i), BL))$
(ii) $Nested(i) = Nested(Prev(bp(i), BL)) = 0$.

4. Each index i, $bp(i) > i$, in BL corresponds to a multi-pseudoknotted loop with $i.bp(i)$ as external base pair if the followings are satisfied: (i) Next$(i, BL) = bp(\text{Prev}(bp(i), BL))$ (ii) $Nested(i) > 0$ or $Nested(Prev(bp(i), BL)) > 0$. Children corresponding to $Tuples(i)$ and $Tuples(Prev(bp(i), BL))$ along with $(Next(i, BL), Prev(bp(i), BL)))$ will be the tuples corresponding to this multi-pseudoknotted loop.

Running Time:
Step 2 can be easily (similar to Merge Sort procedure) done in linear time, in the size of BL and $ChildList$, for each pseudoknotted closed region $i; j$. Then the algorithm needs one scan of BL to identify all inner loops corresponding to $i; j$. So the above algorithm works in linear time, in the size of BL and $ChildList$, for each pseudoknotted closed region $i; j$. Thus, the whole cost is linear in n (number of base pairs in R) plus the number of nodes in the tree, which is at most n. Therefore, finding all inner loops of the structure R is linear in n (number of base pairs in R).

Chapter 6

Akutsu's Algorithm Structure Class

Akutsu [2] introduced simple dynamic programming algorithms for RNA secondary structure prediction. This algorithm is capable of predicting the secondary structure for a restricted class of pseudoknotted RNA structures. We analyse the definition of pseudoknots provided in Akutsu's [2] work and present a concise characterization of the class of structures Akutsu's algorithm can handle.

Akutsu defined pseudoknots in the secondary structure to be either simple or recursive where recursive pseudoknot is a generalized version of simple pseudoknot. We discuss simple pseudoknots in Section 6.1 and recursive pseudoknots (which is equivalent to secondary structure with recursive pseudoknots) in Section 6.3. In Section 6.2 we describe secondary structures with simple pseudoknots which are an intermediate class of structures defined by Akutsu [2].

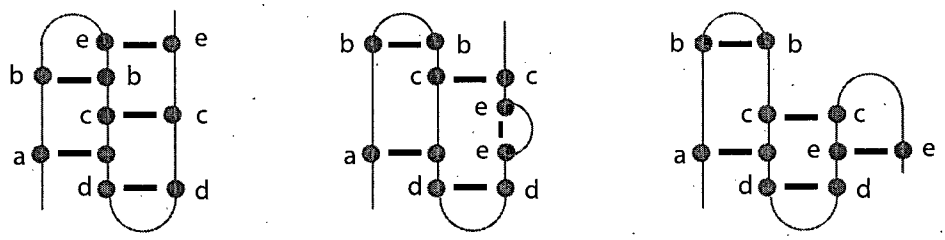
6.1 Simple Pseudoknot Structures

Definition 6.1.1 Akutsu definition for simple pseudoknot:

Let P be a pattern of size $2n$ over an alphabet Σ of size n. P is called a simple pseudoknot *if there exist positions (indices) $j_0 = j_0(P)$ and $j_0' = j_0'(P)$ $(1 < j_0' \leq j_0 \leq 2n)$ for which the following conditions are satisfied:*

- **A1:** *Each $a \in \Sigma$ satisfies either $1 \leq Left(a) < j_0' \leq Right(a) \leq j_0$ or $j_0' \leq Left(a) \leq j_0 < Right(a) \leq 2n$.*
- **A2:** *If a and b, for each $a, b \in \Sigma$, satisfy either $Left(a) < Left(b) < j_0'$ or $j_0' \leq Left(a) \leq Left(b)$, then $Right(a) > Right(b)$ holds.*

The two indices j_0 and j_0' witness that P is a simple pseudoknot *and we refer to them as* witnesses *for P. (Figure 6.1(a))*



(a) (b) (c)

Figure 6.1: (a) Secondary structure with pattern $P = abebcaddce$: Simple pseudoknot with $j_0 = Left(e)$ and $j_0' = Left(d)$ as witnesses. (b) Secondary structure with pattern $P = abbcaddeec$: Recursive pseudoknot as $P = abbcaP_1eec$ where $P_1 = dd$ and $abbcaeec$ are both simple pseudoknot. (c) Secondary structure with pattern $P = abbcaddece$: neither simple or recursive.

According to **A1**, symbols in a *simple pseudoknot* P can be divided in to two groups:

- **G1:** each symbol $a \in G1$ satisfies $1 \leq Left(a) < j_0' \leq Right(a) \leq j_0$.
- **G2:** each symbol $a_i \in G2$ satisfies $j_0' \leq Left(a) < j_0 < Right(a) \leq 2n$.

So it can be derived directly from **A2** that $P \downarrow G1$ is of the form $a_1a_2...a_ra_r...a_2a_1$, where $a_i \in G1, \forall 1 \leq i \leq r$. The same is true for $P \downarrow G2$.

For example, in the structure of Figure 6.1(a), $a, b \in G1$ and $c, d, e \in G2$.

Definition 6.1.2 Our definition for simple pseudoknot:

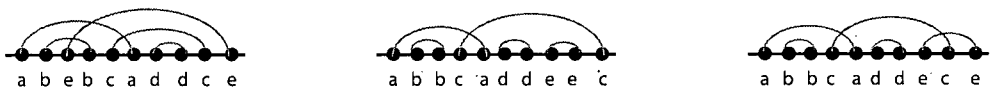
A pattern P is a simplest pseudoknot *if and only if it admits either of these two cases:*

- **B1:** *It is equal to a_1a_1.*
- **B2:** *It is equal to either $a_1a_iP_1a_ia_1P_2$ or $a_1P_1a_ia_1a_iP_2$, where $a_1P_1a_1P_2$ is a* simplest pseudoknot.

A pattern P is a simple pseudoknot *if and only if either it is a* simplest pseudoknot *or it is equal to $a_1P_1a_1a_ia_{i+1}\ldots a_ra_r\ldots a_{i+1}a_iP_2$, for some $a_1, a_i, ...a_r \in \Sigma$, where $a_1P_1a_1P_2$ is a* simplest pseudoknot.

Corollary 6.1.1 *It can be easily infered, by using simple induction on the definition of* simplest pseudoknot, *that there is no* Left *symbol after (on the right side of) the second occurrence of a_1, where a_1 is the first symbol in the pattern.*

Then, it can be easily infered, from the simple pseudoknot *definition, that a* simplest pseudoknot *pattern is equivalent to the* simple pseudoknot *pattern in which there is no* Left *symbol after (on the right side of) the second occurrence of a_1, where a_1 is the first symbol in the pattern.*



(a) (b) (c)

Figure 6.2: Arc diagram representation of structures in Figure 6.1.

6.1.1 Equivalence of Definitions

Theorem 6.1.1 *Definition 6.1.1 and 6.1.2 are equivalent.*

Proof

Part 1 If P satisfies Definition 6.1.1 then P satisfies Definition 6.1.2.

Proof

We use induction to prove this. If P is of length 2 and satisfies Definition 6.1.1 then it is equal to a_1a_1 and so satisfies Definition 6.1.2 (Base case).

Assume that it is true for all P of length $k < l$. Let P, of length l, satisfy Definition 6.1.1. Let $j_0'(P)$ and $j_0(P)$ be witnesses for P. Then either (i) there is a *Left* symbol after the second occurrence of a_1, or (ii) there isn't.

All of the symbols after the second occurrence of a_1 should belong to **G2**. So in case(i), these symbols form a $a_ia_{i+1}\ldots a_ra_r\ldots a_{i+1}a_iP_2$ pattern, where P_2 only consists of *Right* symbols. Therefore $P = a_1P_1a_1a_ia_{i+1}\ldots a_ra_r\ldots a_{i+1}a_iP_2$. If we let $P' = a_1P_1a_1P_2$, let $j_0(P') = Right(P', a_1)$ and let $j_0'(P') = j_0'(P)$ then both conditions **A1** and **A2** are still satisfied for P' with $j_0'(P')$ and $j_0(P')$ as witnesses. So, $P' = a_1P_1a_1P_2$ satisfies Definition 6.1.1 and therefore, by induction, it satisfies Definition 6.1.2 and in fact must be a simplest pseudoknot, since P_2 only contains *Right* symbols (Corollary 6.1.1). Therefore P satisfies Definition 6.1.2.

Now consider case(ii) where there is no *Left* symbol after the second occurrence of a_1. The symbol a_i, right before the second occurrence of a_1, either belongs to **G1** or **G2**. To satisfy Definition 6.1.1 it must be that (i) $P = a_1a_iP_1a_ia_1P_2$, if $a_i \in G1$, or (ii) $P = a_1P_1a_ia_1a_iP_2$, if $a_i \in G2$, while P_2 only consists of *Right* symbols which definitely belong to **G2**. In both cases $j_0(P) = Right(P, a_1)$ and $j_0' = Right(P, a_1) - 1$ if $P_1 = \epsilon$ or $j_0'(P) < Right(P, a_1) - 1$ otherwise. If we let $P' = P \downarrow a_i$, let $j_0(P') = Right(P', a_1)$ and make $j_0'(P')$ to be equal to either (1) $Right(P', a_1)$ if $P_1 = \epsilon$, or (2) $j_0'(P)$ if $P_1 \neq \epsilon$ and $a_i \in G2$, or (3) $j_0'(P) - 1$ if $P_1 \neq \epsilon$ and $a_i \in G1$, then both conditions **A1** and **A2** are still satisfied for P' with $j_0'(P')$ and $j_0(P')$ as witnesses. So, $P' = a_1P_1a_1P_2$ satisfies Definition 6.1.1 and therefore, by induction, it satisfies Definition 6.1.2 and in fact must be a simplest pseudoknot, since P_2 only contains *Right* symbols (Corollary

6.1.1). So, by *simplest pseudoknot* definition, P is a simplest pseudoknot and therefore by Corollary 6.1.1 P satisfies Definition 6.1.2.

□

Part 2 If P satisfies Definition 6.1.2 then P satisfies Definition 6.1.1.

Proof

We use induction to prove this. If P is of length 2 and satisfies Definition 6.1.2 then it is equal to a_1a_1 and so satisfies Definition 6.1.1 (Base case).

Assume that it is true for all P of length $k < l$. Let P, of length l, satisfy Definition 6.1.2.

Suppose that either $P = a_1a_iP_1a_ia_1P_2$ or $P = a_1P_1a_ia_1a_iP_2$ and $P' = a_1P_1a_1P_2$ is a *simplest pseudoknot.* By Corollary 6.1.1 and using induction, P' satisfies Definition 6.1.1. Let $j_0'(P')$ and $j_0(P')$ be witnesses for P', so the following are true: (i) $j_0(P') = Right(P', a_1)$, and (ii) $j_0' = Right(P', a_1)$ if $P_1 = \epsilon$ or $j_0'(P')$ is an index of a symbol in P_1 otherwise. If we let $j_0(P) = Right(P, a_1)$ and let $j_0'(P)$ to be equal to either (1) $Right(P, a_1) - 1$ if $P_1 = \epsilon$, or (2) $j_0'(P')$ if $P_1 \neq \epsilon$ and $a_i \in G2$, or (3) $j_0'(P') + 1$ if $P_1 \neq \epsilon$ and $a_i \in G1$, then both conditions **A1** and **A2** are true for P and therefore P satisfies Definition 6.1.1.

Finally, suppose that $P = a_1P_1a_1a_ia_{i+1} \ldots a_ra_r \ldots a_{i+1}a_iP_2$ and $P' = a_1P_1a_1P_2$ is a *simplest pseudoknot.* By Corollary 6.1.1 and using induction, P' satisfies Definition 6.1.1. Let $j_0'(P')$ and $j_0(P')$ be witnesses for P', so the following are true: (i) $j_0(P') = Right(P', a_1)$ (ii) $j_0' = Right(P', a_1)$ if $P_1 = \epsilon$ or $j_0'(P')$ is an index of a symbol in P_1 otherwise. If we let $j_0(P) = Left(P, a_r)$ and $j_0'(P) = j_0'(P')$ then both conditions **A1** and **A2** are true for P and therefore P satisfies Definition 6.1.1.

□

□

6.1.2 Simple Pseudoknot Membership Test: Linear Time Algorithm

The algorithm for testing whether a pattern P is simple pseudoknot has two steps. In the first step it deals with the $a_ia_{i+1} \ldots a_ra_r \ldots a_{i+1}a_i$ subpattern and removes it from P make the pattern a *simplest pseudoknot.* This can be done by scanning the symbols in the pattern starting from the symbol after the second occurrence of a_1 and doing the following:

1. While the current symbol, a_i, is a *Left* symbol push it on the top of the stack ST (which is empty at the begining) and move to the next symbol.

2. While the current symbol, a_i, is a *Right* symbol and we have a_i on the top of the stack, pop a_i from top of the stack and move to the next symbol.

3. If the stack ST is empty remove all of the symbols between the second occurrence of a_1 and the current symbol from P.

Running time:

It can be easily seen that the above algorithm works in linear time as it checks each symbol at most once.

Next we should figure out if P is simplest pseudoknot. We use both cases in the definition of *simplest pseudoknot* to construct the desired algorithm.

We define two *simplify* operations according to **B2**:

i $a_1 a_i S_1 a_i a_1 S_2$ is converted to $a_1 S_1 a_1 S_2$.

ii $a_1 S_1 a_i a_1 a_i S_2$ is converted to $a_1 S_1 a_1 S_2$.

We define one more operation, *final* operation according to **B1**:

iii $a_1 a_1$ is converted to ϵ.

In these cases we say that a simple/final operation is applicable to a_1.

The algorithm for testing whether the pattern P is a *simplest pseudoknot* works as follows:

1. While one of the *simplify* operations, **i** or **ii**, is applicable on the first symbol,a_1, apply it.
2. Do the *final* operation, **iii**, on a_1 if it is applicable.
3. Return true if the pattern is empty and false otherwise.

For example, pattern $P = abebcaddce$ corresponding to the structure in Figure 6.1(a) is a simple pseudoknot pattern as we can simplify it as the following: (1) $abebcaddce \rightarrow abebcace$ by removing dd in the first step, (2) $abebcace \rightarrow abebae$ by (ii), (3)$abebae \rightarrow aeae$ by (i), (4) $aeae \rightarrow aa$ by (ii), and finally (5) $aa \rightarrow \epsilon$ by (iii).

Simple test running time:

Each *simplify* operation takes constant time, and the algorithm does at most n (number of distinct symbols in P) *simplify* operation on the pattern. Therefore the running time of the algorithm will be $O(n)$.

So the whole test can be done in linear time.

6.2 Secondary Structures with Simple Pseudoknots

Definition 6.2.1 *A pattern P is called a* pattern with simple pseudoknots *if and only if for some strings $S_1, P_1, S_2, P_2, ..., S_t, P_t, S_{t+1} \in \Sigma^*$:*

- $P = S_1 P_1 S_2 P_2 \ldots S_t P_t S_{t+1}$,

- *each* P_i, $1 \leq i \leq t$, *is a* simple pseudoknot, *and*
- $P' = S_1 S_2 \ldots S_t S_{t+1}$ *is a* pseudoknot free *pattern.*

Note that a pseudoknot free pattern corresponds to the case of $t = 0$ *and qualifies as a pattern with simple pseudoknots.*

We say that a secondary structure R *is a* secondary structure with simple pseudoknots *if its corresponding pattern,* P, *is a pattern with simple pseudoknots.*

We give a characterization of secondary structure with simple pseudoknots in Theorem 6.2.1. First we prove two useful lemmas.

Lemma 6.2.1 *If pattern* P *is a* simple pseudoknot *then it is a* closed pattern.

Proof Let pattern $P = a_1 ... a_m$ be a *simple pseudoknot.* To prove that P is a *closed pattern* it is enough to prove that its corresponding structure, $[Left(a_1); Right(a_m)]$, is a *closed* region.

Clearly $[Left(a_1); Right(a_m)]$ is a *weakly* closed region. Assume that it is not *closed.* Then there is a_r, $a_r \neq a_1$ and $a_r \neq a_m$, such that either $[Left(a_1); Right(a_r)]$ or $[Left(a_r); Right(a_m)]$ is *weakly* closed. In the former case, $Right(a_r)$ should be greater than $Right(a_1)$ which means that a_r belongs to **G2**. Therefore $Right(a_r) > Left(a_m)$ and we also have $Right(a_r) < Right(a_m)$. Then we can conclude that $[Left(a_1); Right(a_r)]$ is not *weakly* closed. In the latter case, $Left(a_r)$ should be less than $Left(a_m)$ which means that a_r belongs to *First group.* Therefore $Left(a_r) < Right(a_1)$ and we also have $Left(a_r) > Right(a_1)$. Then we can conclude that $[Left(a_r); Right(a_m)]$ is not *weakly* closed. (Contradiction)

□

Lemma 6.2.2 *Any closed region of a simple pseudoknot structure* R *of length* n *is either* $[1; n]$ *or is pseudoknot free.*

Proof

Type 1: regions $[i; j]$ with $i < j'_0$ and $j \geq j_0$. Suppose that $1.i'$ and $j'.n$. By Definition 6.1.1 $j'_0 \leq i'$ and $j' \leq j_0$, so both i' and j' are in $[i; j]$. Therefore 1 and n should be in $[i; j]$ which means that $[i; j] = [1; n]$. So this type only contains one region which is the whole structure R.

Type 2: regions $[i; j]$ such that either: (i) $i < j'_0$ and $j \leq j_0$ (where clearly $j'_0 \leq j$), or (ii) $j'_0 \geq i \leq j_0$ and $j_0 < j$. Then it is easy to see that $i.j$. So this type of closed region is pseudoknot free.

□

Theorem 6.2.1 *A pattern P is a* pattern with simple pseudoknots *if and only if all of the closed patterns corresponding to the nodes in its parse tree $T(P)$ are either pseudoknot free or* simple pseudoknot.

Proof

To prove our theorem, we need to prove the following.

Part 1 If for all of the nodes V in parse tree $T(P)$ one of the following is true, then P is a *pattern with simple pseudoknots* : (i) the pattern corresponding to V is pseudoknot free, or (ii) the pattern corresponding to V is a *simple pseudoknot.*

Proof

Assume that each node V in $T(P)$ either corresponds to a pseudoknot free closed pattern or its corresponding pseudoknotted closed pattern is a *simple pseudoknot.* By Lemma 6.2.2 there are no two nodes V and U such that their corresponding pseudoknotted closed patterns are *simple pseudoknots* and one of them is the ancestor of another. Assume that $P_1, ..., P_t$ are the closed patterns corresponding to the pseudoknotted nodes, ordered by the left index of hte closed pattern. So we can write $P = S_1P_1S_2P_2 \ldots S_tP_tS_{t+1}$ where each P_i, $1 \leq i \leq t$, is a *simple pseudoknot* pattern and $S_1S_2 \ldots S_tS_{t+1}$ is a *pseudoknot free* pattern. Therefore P is a *pattern with simple pseudoknots.*

□

Part 2 If P is a *pattern with simple pseudoknots* then one of the following is true for any node V in $T(P)$: (i) the pattern corresponding to V is pseudoknot free, or (ii) the pattern corresponding to V is a *simple pseudoknot.*

Proof

Assume that P is a *pattern with simple pseudoknots.* So $P = S_1P_1S_2P_2 \ldots S_tP_tS_{t+1}$ where each P_i, $1 \leq i \leq t$, is a *simple pseudoknot* pattern and $S_1S_2 \ldots S_tS_{t+1}$ is a *pseudoknot free* pattern.

Each P_i, $1 \leq i \leq t$ is a *closed pattern* (by Lemma 6.2.1) and so a node in $T(P)$ is associated with it. The other nodes in the tree should correspond to pseudoknot free closed patterns. So closed regions corresponding to all nodes V are either pseudoknot free or *simple pseudoknot.*

□

□

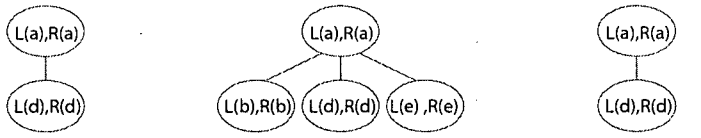
6.2.1 Secondary Structure with Simple Pseudoknots Membership Test

Therefore the following algorithm can easily test whether a secondary structure R, with pattern representation P (Lemma 6.2.3), is a *secondary structure with simple pseudoknots*:

1. Construct the parse tree $T(P)$.

2. Starting from the root, check the nodes in each level of $T(P)$. If the simple pseudoknot test returns *true* on all of the pseudoknotted closed patterns corresponding to nodes in $T(P)$ then return *true* and return *false* otherwise.

For example, the structure in Figure 6.1(a) is a secondary structure with simple pseudoknots. The only pseudoknotted node in its parse tree (Figure 6.3(a)) is the root node whose pattern is $P = abebcaddce$ and P is a simple pseudoknot. But, the structure in Figure 6.1(b) is not a secondary structure with simple pseudoknots. The pattern $P = abbcaddeec$ corresponding to its root node (Figure 6.3(b)) is not a simple pseudoknot.



(a) (b) (c)

Figure 6.3: Parse trees for the structures in Figure 6.1.

Simple structure test running time:

Constructing the parse tree can be done in $O(n)$ running time, n = number of base pairs in the structure, (Lemma 6.2.3). Then all we need is to run the simple pseudoknot test on all of the pseudoknotted nodes in the tree. According to Lemma 6.2.2, if the closed region corresponding to a node V is a *simple pseudoknot*, then none of the nodes in the subtree rooted at node V is pseudoknotted, and therefore the algorithm does not need to run the simple pseudoknot test on them. So the whole simple pseudoknot test on the nodes of $T(R)$ is linear in the number of base pairs and the running time of the algorithm is $O(n)$.

Lemma 6.2.3 *We can obtain all the patterns and the private patterns corresponding to closed regions of structure R easily in linear time. This also follows that we can obtain $T(P)$ from $T(R)$ in linear time.*

Proof Consider the following algorithm which assigns a symbol to each paired base index in L, the *linked list representation* of R: (i) let $i = 1$, (ii) scan L from left to right, when $l < bp(l) \neq 0$ is scanned assign symbol a_i to both l and $bp(l)$ and increase i.

It is easy to see that the above algorithm is linear in the length of L (and therefore in the length of R). Having L with symbol assignments and $T(R)$ we can easily figure out the pattern corresponding to each closed region in R , and therefore $T(P)$, in linear time in n (number of base pairs in R).

To obtain the private pattern PC corresponding to a closed region C, we just need to remove the patterns corresponding to C's nested closed regions from PC. The pattern corresponding to nested closed region $i';j'$ can be removed in constant time by making Prev(i', L) and Next(j', L) to be adjacent (by updating the links). So this can be done in linear time in the number of nested closed region in C. Therefore the cost for the whole closed regions is linear in the number of closed region in R (number of nodes in the tree) which is at most n (number of base pairs in R).

So we can obtain all patterns and private patterns corresponding to closed regions of R in linear time in n (number of base pairs in R).

□

6.3 Secondary Structures with Recursive Pseudoknots

Definition 6.3.1 *A pattern P is called a* recursive pseudoknot *if and only if P is a* simple pseudoknot *or $P = P_1P_2P_1'$ where P_2 is a nonempty* simple pseudoknot *and P_1P_1' is a* recursive pseudoknot.

We say that an RNA secondary structure R is a secondary structure with recursive pseudoknots *or conveniently* recursive pseudoknot structure *if its corresponding pattern P is a* recursive pseudoknot.

Claim 6.3.1 *Given the above definition we claim that the pattern P is a recursive pseudoknot if and only if all of the private patterns corresponding to the nodes in $T(P)$ are simple pseudoknot.*

Proof

The simple pseudoknot test returns true on a node if its corresponding private pattern is a *simple pseudoknot.* So to prove our claim we need to prove the following.

Part 1 If the simple pseudoknot test returns *true* on all nodes in $T(P)$, then P is a *recursive pseudoknot.*

Proof

We prove it by using induction on the size of P, $2n$. It is clearly true for $2n = 2$ (Base case). Let assume that it is true for all patterns of size $2l < 2n$ (Induction Hypothesis). We prove that it is true for any pattern P of length $2n$.

Let L_1 be one of the leaf nodes in $T(P)$. Assume that simple pseudoknot test returns *true* on L_1 which means that its corresponding private pattern C_1 is a simple pseudoknot. Either L_1 is the only node in the tree (which means that $C_1 = P$) or there are more nodes in the tree than L_1.

In the former case, since P is simple pseudoknot, by Definition 6.3.1 P is a recursive pseudoknot.

In the latter case $P = P_1 C_1 P_1'$. By eliminating L_1 from $T(P)$ we can get the parse tree, T_1, corresponding to $P_1 P_1'$. Assume that the simple pseudoknot test returns *true* on all of the nodes in $T(P)$ which means that it returns *true* on all of the nodes in T_1. Using induction $P_1 P_1'$ is recursive pseudoknot. We saw before that C_1 is a nonempty simple pseudoknot. Therefore, by the definition of recursive pseudoknot $P = P_1 C_1 P_1'$ is a *recursive pseudoknot.*

□

Part 2 If P is a *recursive pseudoknot* then the simple pseudoknot test returns *true* on all of the nodes in $T(P)$.

Proof

We prove it by using induction on the size of P, $2n$. It is clearly true for $2n = 2$ (Base case). Let assume that it is true for all patterns of size $2l < 2n$ (Induction Hypothesis). We prove that it is true for any pattern P of length $2n$.

Assume P is a *recursive pseudoknot.* Then either P is a *simple pseudoknot* or it is of the form $P_1 P_2 P_1'$ where P_2 is a nonempty *simple pseudoknot* and $P_1 P_1'$ is a *recursive pseudoknot.*

In the former case the simple pseudoknot test should return *true* on all of the nodes in $T(P)$. (Using Lemma 6.3.1).

In the latter case assume that T_2 and T_1 are respectively the parse trees for $P_1 P_2 P_1'$ and P_2. It is easy to see that $T(P)$ can be obtained by concatenating T_1 and T_2. (This is done by finding a node in T_1 which should be the parent of the root node in T_2 and making the parent-child relationship between them.) Using induction, the simple pseudoknot test should return *true* on all of the nodes in T_1 and T_2 and therefore on all of the nodes in $T(P)$. This makes the proof complete.

□

□

Lemma 6.3.1 *If pattern P is a* simple pseudoknot *then the simple pseudoknot test returns* true *on all of the nodes in $T(P)$.*

Proof

We use induction to prove this. It is true for $2n = 2$ (Base case). Assume that it is true for all structures of size $2k < 2n$ (Induction Hypothesis). We prove that it is true for any structure of size $2n$.

If $T(P)$ consists of just one node then this node should return *true.* Assume that $T(P)$ has more than one node. By Lemma 6.2.2 it should have some nodes of the second type, **Type 2**. As the **Type 2** nodes correspond to the pseudoknot free closed regions then there should be a leaf node $i.j$ such that either $j_0' = j$ or $j_0 = j$. It is clear that $i.j$

is a simple pseudoknot, and it is easy to see that by removing $i.j$ from R, the remaining structure is still a simple pseudoknot. So by induction, the simple pseudoknot test returns *true* on all of the other nodes in $T(P)$.

□

6.3.1 Recursive Pseudoknot Membership Test

Now we can easily test whether a secondary structure R, with pattern representation P (Lemma 6.2.3), is a *recursive pseudoknot structure* by the following algorithm:

1. Construct the parse tree $T(P)$.
2. If the simple pseudoknot test returns *true* on all of the private patterns corresponding to the nodes in $T(P)$ then return *true* and return *false* otherwise.

For example, the structure in Figure 6.1(b) is a recursive pseudoknot. All private patterns corresponding to the nodes in its parse tree (Figure 6.3(b)) which includes *acac*, *aa*, *bb* and *ee* are simple pseudoknots. But the structure in Figure 6.1(c) is not a recursive pseudoknot. The private pattern corresponding to its root node (Figure 6.3(c)) which is *abbcaece* is not a simple pseudoknot.

Recursive structure test running time:

Constructing the parse tree can be done in $O(n)$ running time, n = number of base pairs in the structure, (Lemma 6.2.3). Then all we need is to run simple pseudoknot test on all of the private patterns corresponding to the nodes in the tree (Lemma 6.2.3). The simple pseudoknot test is linear (in the number of base pairs) too, so the running time of the whole algorithm is $O(n)$.

6.4 Classification of Biological Structures

Condon et al.[5] provide a simple characterization of the classes of structures handled by Rivas and Eddy (R&E) [13], Dirks and Pierce (D&P) [7], Lyngso and Pederson (L&P) [11] and Akutsu [2] algorithms. They also provide a linear time algorithm to test if an input structure is in the R&E, D&P, and L&P classes. They applied their algorithms to classify biological structures from PseudoBase (PBase) [16], the Nucleic Acids Database (NDB) [3], 16S and 23S ribosomal RNA and Group I and Group II Introns from Gutell Database [4].

We also applied our algorithms for Akutsu class membership, for both secondary structures with simple pseudoknots (Section 6.2) and secondary structures with recursive pseudoknots (Section 6.3), on the same biological structures. Table 6.1 presents our results and also the results reported by Condon et al.[5].

As reported by Condon et al.[5], the R&E structure class is indeed very general, whereas the L&P class misses almost all of the 16s rRNA structures. The Akutsu Simple

	PBase	16S	23S	Gp I Intron	Gp II Intron	NDB
# Strs	240	152	69	10	3	12
Avg. #Bps	14.2	466	763.1	128.9	209	312.4
PKF	0	0	14	0	0	1
AkutsuS	204	0	14	0	0	1
L&P	231	12	14	10	0	1
D&P	232	150	14	10	0	5
AkutsuR	232	150	14	10	0	5
R&E	240	152	25	10	0	7

Table 6.1: Structure classification. Columns 2-7 present data for each RNA data set. For each data set (column), the entry in first row lists the number of structures in the data set. The second row lists the average number of base pairs in the structures. The remaining rows list the number of structures of the data set that are in the PKF, Akutsu(secondary structures with simple pseudoknots), L&P, D&P, Akutsu(secondary structures with recursive pseudoknots) and R&E classes, respectively.

class behaves worse than L&P, as it misses all of the structures in Group I additionally. Although it is proved theoretically that Akutsu Recursive class is more general than D&P class, the test result shows that they can both handle the same class of given biological structures.

Chapter 7

Conclusion and Future Work

RNA molecules play diverse roles in the cell. The structure of RNA molecules is often the key to their function and predicting the secondary structure of RNA molecules is thus an important problem in computational molecular biology. Algorithms for computational prediction of RNA secondary structures, which find the minimum free energy (mfe) structure are widely used, but experiments show that existing polynomial time algorithms are not so powerful in predicting the base pairs in known biological RNA structures (experimental structures) correctly. Therefore, improving existing algorithms or designing new algorithms, which are more capable in predicting the real secondary structure of RNA sequences, is an important challenge.

Towards this goal, in this thesis, we presented a precise definition of the structural elements in a secondary structure, and also a comprehensive way to classify the type of loops that arise in pseudoknotted structure (Chapter 2). Then we introduced a parsing algorithm which parses a given secondary structure to its structural components (Chapter 4). This parsing algorithm, along with the classification scheme for the loops in a pseudoknotted secondary structure, can be used in analysing existing prediction algorithms and known biological RNA structures in order to design new, more powerful, prediction algorithms.

Our parsing algorithm can be used in other applications. In this thesis we presented two of them: (i) calculating the free energy of a given secondary structure in linear time (Chapter 5), and (ii) constructing a linear time algorithm to test whether the prediction algorithm introduced by Akutsu [2] can handle a given structure (Chapter 6).

Our work can be continued in future in several directions:

- Heuristic algorithms commonly use a procedure to calculate the free energy for a given sequence and structure. Incorporating the linear free energy calculation algorithm (presented in Chapter 5), into heuristic algorithms may cause improvements in their efficiency.

- The parsing algorithm can be used in order to analyse known biological RNA structures, in order to find out what structures occur more frequently in biology.

- The method for parsing a secondary structure appears to yield a linear time algorithm for finding the maximum independent set in a circle graph. The previous best algorithm had running time $O(n \log n)$ [1].
- The parsing algorithm plus extra functions presented in energy calculation part seems to be useful in calculating the minimum number of base pairs that need to be removed in order to make a secondary structure pseudoknot free.
- The classification model we presented in this thesis can be studied and improved if needed. This wasn't an easy task before, but now as we have some new notions and conventions it can be done more easily.
- There is no precise characterization of the class of structures handled by Uemura et al.[15] algorithm (Uemura). Our work may be useful in providing such a characterization, and also constructing an algorithm to test whether Uemura's algorithm can handle a given structure.

Bibliography

[1] Apostolico, A., Mikhail J. Atallah, Susanne E. Hambrusch. 1992. New clique and independent set algorithms for circle graphs. *Discrete Applied Mathematics* 36: 1-24.

[2] Akutsu, T. 2000. Dynamic programming algorithms for RNA secondary structure prediction with pseudoknots. *Discrete Applied Mathematics* 104:45–62.

[3] Berman, H. M. et al. 1992 The Nucleic Acid Database: A comprehensive relational database of three-dimensional structures of nucleic acids. *Biophys.J.* 63:751-759.

[4] Cannone J.J. et al. 2002. The Comparative RNA Web (CRW) site: An online database of comparative sequence and structure information for ribosomal, intron and other RNAs. *BioMed Central Bioinformatics* 3:15. [Correction: BioMed Central Bioinformatics. 3:15.]

[5] Condon, A., Davy, B., Rastegari, B., Zhao, S. and Tarrant, T. 2004. Classifying RNA pseudoknotted structures. *Theoretical Computer Science* to be appear?.

[6] Dennis, C. 2002. The brave new world of RNA. *Nature* 418(11):122-124.

[7] Dirks, R. M. and Pierce, N. A. 2003. A partition function algorithm for nucleic acid secondary structure including pseudoknots. *J. Comput. Chem.* 24(13):1664-1677.

[8] Gultyaev, A.P., Van Batenburg, F.H.D. and Pleij, C. W. A. 1995. The computer simulation of RNA folding pathways using a genetic algorithm. *Journal of Molecular Biology* 250: 37–51.

[9] Gultyaev, A.P., Van Batenburg, F.H.D. and Pleij, C. W. A. 1999. An approximation of loop free energy values of RNA H-pseudoknots. *RNA* 5: 609-617.

[10] Haslinger, M.C. 2001. Prediction algorithms for restricted RNA pseudoknots.

[11] Lyngsø R. B. and Pedersen, C. N. 2000. RNA pseudoknot prediction in energy-based models. *J. Computational Biology* 7(3):409–427.

[12] Pleij, C.W.A. 1990. Pseudoknots: a new motif in the RNA game. *Trends Biochem Science* 15(4): 143-147.

[13] Rivas E. and Eddy, S. R. 1999. A dynamic programming algorithm for RNA structure prediction including pseudoknots. *J. Molecular Biology* 285:2053–2068.

[14] Ruan, J., Stormo, G.D. and Zhang, W. 2004. Iterative Loop Matching (ILM) an approach to the prediction of RNA secondary structures with pseudoknots. *Bioinformatics* 20-1: 58–66.

[15] Uemura, Y., Hasegawa, A., Kobayashi, S. and Yokomori, T. 1999. Tree adjoining grammars for RNA structure prediction. *Theoretical Computer Science* 210:277–303.

[16] Van Batenburg, F.H.D. , Gultyaev, A.P., Pleij, C. W. A., Ng, J. and Oliehoek J. 2000. Pseudobase: a database with RNA pseudoknots. *Nucl. Acids Res* 28(1):201-204.