Dynamic Programming based RNA Pseudoknot Alignment

Mathias Möhl

Saarbrücken 2009

Dissertation zur Erlangung des Grades
des Doktors der Ingenieurwissenschaften
der Naturwissenschaftlich-Technischen Fakultät
der Universität des Saarlandes

Berichterstatter:
Prof. Dr. Gert Smolka
Prof. Dr. Rolf Backofen
Prof. Dr. Hans-Peter Lenhof

Dekan:
Professor Dr. Joachim Weickert

Prüfungsausschuss:
Prof. Dr. Raimund Seidel
Prof. Dr. Gert Smolka
Prof. Dr. Rolf Backofen
Prof. Dr. Hans-Peter Lenhof
Dr. Eyad Alkassar

Tag des Kolloquiums:
11. Februar 2010

Abstract

Pseudoknots are certain structural motifs of RNA molecules. In this thesis we consider the problem of RNA pseudoknot alignment. Most current approaches either discard pseudoknots in order to be efficient or rely on heuristics generating only approximate solutions. This work focuses on dynamic programming based alignment methods and proposes two new approaches for an exact solution of the alignment problem in the presence of pseudoknot structures. The first approach is able to handle arbitrary pseudoknots, however, does not guarantee a polynomial runtime for all instances, due to the NP-hardness of the problem. Nevertheless, an analysis in terms of parameterized complexity shows that the algorithm is fixed parameter tractable for a parameter that is small in practice. The second approach is a general scheme for the alignment of restricted classes of pseudoknots in polynomial time. It is motivated by existing RNA pseudoknot prediction algorithms. We show how to embed seven of those algorithms in a common scheme and present an analogous scheme for the alignment problem, which yields for each of the structure prediction algorithms a corresponding alignment algorithm. The alignment algorithms handle the same class of pseudoknots as the corresponding prediction algorithms and the time and space complexity is only increased by a linear factor, compared to the respective prediction algorithm. Both approaches have been implemented to evaluate their applicability in practice.

Zusammenfassung

In dieser Dissertation beschäftige ich mich mit dem Alignment von bestimmten RNA Strukturen, die als Pseudoknoten bezeichnet werden. Da dieses Problem NP-hart ist, berücksichtigen die meisten bisher verfügbaren Alignmentverfahren um effizient zu sein entweder keine Pseudoknoten oder berechnen nur approximierte Lösungen mit Hilfe von Heuristiken. In der vorliegenden Arbeit beschreibe ich zwei neue Verfahren, die mit Hilfe von dynamischer Programmierung eine exakte Lösung für das Alignmentproblem von Pseudoknotenstrukturen berechnen. Das erste Verfahren kann beliebige Pseudoknoten alignieren und hat, da es sich hierbei um ein NP-hartes Problem handelt, im allgemeinen keine polynomiell beschränkte Laufzeit. Eine parametrische Komplexitätsanalyse zeigt allerdings, dass der Algorithmus parametrisierbar (fixed parameter tractable) in Bezug auf einen in der Praxis kleinen Parameter ist. Das zweite Verfahren ermöglicht es, unterschiedliche eingeschränkte Klassen von Pseudoknoten in polynomieller Zeit zu alignieren. In einem ersten Schritt zeige ich hierzu, wie man existierende Vorhersagealgorithmen für sieben solcher Klassen in ein gemeinsames Schema einbetten kann. Dann entwickele ich ein analoges Schema für das Alignment von Pseudoknoten, das zu jedem der Vorhersagealgorithmen einen entsprechenden Alignmentalgorithmus mit nur linear erhöhter Speicher- und Zeitkomplexität liefert. Beide Verfahren wurden auch implementiert um die Praxistauglichkeit zu evaluieren.

Acknowledgments

First of all, I want to thank Gert Smolka and Rolf Backofen for the supervision of my thesis. Gert granted me the freedom to focus my research on the topics I am interested in and to work with the people and at places where I could be most productive. In particular he initiated the collaboration with Rolf's Bioinformatics Group in Freiburg and made it possible for me to spend much time there. Rolf advised me with his expert knowledge in all kinds of questions, ranging from technical details to the general focus of the work. His enthusiasm for the topic was a constant source of motivation for me. I am also grateful to Hans-Peter Lenhof for reviewing the thesis.

Additionally, I would like to take the opportunity to thank my current and former colleagues both at the Programming Systems Lab in Saarbrücken and at the Bioinformatics Chair in Freiburg for the pleasant atmosphere, fruitful discussions, and their support. In Saarbrücken these are Gert, Chad Brown, Ralph Debusmann, Mark Kaminski, Marco Kuhlmann, Sandra Neumann, Andreas Rossberg, Jan Schwinghammer, Guido Tack, and Ann van de Veire. In Freiburg, in particular I want to mention Sebastian Will who gave essential inspirations leading to major contributions of this thesis. The countless discussions with him and Rolf where always exciting, productive, and moreover very enjoyable. For me, this way of brainstorming, sharing, and refining ideas is the core of what defines scientific work and it is priceless to find people that are on the same wavelength so that results emerge that go far beyond what I could have achieved alone. I in particular enjoyed the friendly atmosphere among the lab members in Freiburg which are, besides Rolf and Sebastian, Anke Busch, Monika Degen-Hellmuth, Steffen Heyne, Oliver Krieg, Sita Lange, Daniel Maticzka, Andreas Richter, Stefan Jankowski, Kousik Kundu, Martin Mann, and Rileen Sinha. I was surprised by the number of spontaneous coffee and cake breaks and even more by the fact that they are increasing instead of hindering the productivity of the team.

I am also grateful to Markus Bauer, Gunnar Klau and Knut Reinert, who provided me a tailored version of their lara program such that I could compare it to the approaches developed in this thesis.

During the work on this thesis I was funded by the German Research Foundation (DFG) as a member of the graduate studies programme "Quality Guarantees for Computer Systems". I am grateful for the funding which also included a travel grant that allowed me to visit conferences and to travel frequently between Saarbrücken and Freiburg.

Finally, I am grateful to my family, in particular to my parents. Their endless support ranged from encouraging me to start the thesis to proofreading it.

Contents

1 Introduction

1.1 Motivation

The central dogma of molecular biology states that the genetic information of all living organisms is contained in DNA (Deoxyribonucleic acid) that is transcribed into RNA (Ribonucleic acid) and finally translated into amino acid sequences that form proteins. First stated by Francis Crick in 1958 [14] and reformulated by him in a Nature article from 1970 [15], this insight has built the ground for all subsequent research in molecular biology. The development and large scale application of DNA sequencing methods, most prominently the Human Genome Project started in 1990, made huge amounts of the genetic information available. Today, high-throughput sequencing technologies are commonly applied and relatively cheap, but determining the genes in the sequenced DNA and determining the function of the protein that the genes code for is still a much more involved process. While the Human Genome Project had sequenced the entire human genome in 2001, the ENCODE project (ENCyclopedia Of DNA Elements) which started in 2003 with the goal to identify and catalogue the functional elements encoded in the human genome, had covered only 1% of the human genome by the end of their pilot phase in 2007 [12].

For those tasks often comparative methods are used. They are based on the observation that functionally relevant sequences are more conserved during evolution than others. This is due to the fact that a mutation in a functional relevant DNA might result in a loss of function which is prevented by selection pressure. The basis of comparative methods is usually a sequence alignment which compares sequences and helps to identify conserved parts. Comparative methods are also interesting because a similarity in structure often coincides with a similarity in function.

Besides their use for DNA sequence analysis, comparative methods have regained much attention in the last years for RNA analysis. This is due to the discovery of different classes of so called non-protein-coding RNAs, i.e. RNA molecules that do not act as an intermediate between DNA and proteins, but instead perform their own distinctive function. In 2002, the Science Magazine hailed these discoveries as the "Breakthrough of the Year" [13] which dramatically changed the perspective on RNA molecules: before, they where assumed to be only necessary to transport the genetic code from the nucleus to the ribosomes where corresponding proteins are synthesized, now they turned out to be key players of central regulatory and catalytic functions [33, 34].

The motivation of my work presented in this thesis is the development of new, improved algorithms for comparative RNA analysis. I focus on the alignment of RNA structures which is the central step of usual comparative approaches. Compared to

DNA molecules which form a double helix, RNA molecules are able to form complex secondary structures. In RNA alignment it is desirable to compare the molecules not just at their sequence level, but also based on this secondary structure. As mentioned above, comparative methods rely on the fact that evolutionary pressure preserves functionally relevant parts. Since the function is more related to the RNA's secondary structure than to its pure sequence, an alignment at the level of secondary structure usually yields better results than a pure sequence alignment. The drawback of aligning secondary structures is that it is a complex and computationally intensive process. If no structural restrictions are imposed, the problem is NP-hard for the usual scoring schemes [20, 27]. In order to be feasible, most existing approaches are therefore restricted to structures that do not contain so called pseudoknots.

However, already in 1985 Pleij et al. [42] reported that they "have shown that pseudoknots are present in RNA molecules". Recent progress in RNA research shows that pseudoknots are not at all a rare event and, furthermore, are often functionally relevant. Staple and Butcher state in [53] that "among the most prevalent RNA structures is a motif known as the pseudoknot" and later on that pseudoknots "play a variety of diverse roles in biology". Nevertheless, also in experimental RNA research only little work has been done on pseudoknot structures. This is also due do a lack of comparative structure prediction methods. The availability of such methods for pseudoknot free structures has initiated much experimental work on RNA structures, but for pseudoknot structures almost no comparative methods are available so far. In my work I hence address the task to align RNA structures that include pseudoknots.

On the theoretical side the main goal is to thoroughly understand the structure of this problem and in particular where its complexity comes from. This allows to identify the circumstances under which the problem can still be solved efficiently. One option that I explore here is to restrict the class of considered pseudoknots in several ways. I develop alignment algorithms that have a well-balanced trade-off between their computational complexity and the class of pseudoknots that they are able to handle. Besides these algorithms for restricted classes, I also investigate the full problem to align arbitrary pseudoknots. Due to the NP-hardness of this problem there is little hope that it can be solved efficiently for all pseudoknots. Instead, the complexity of the algorithm I develop for arbitrary pseudoknots increases in relation to the structural complexity of the pseudoknots. I formally analyze this relation by means of parametrized complexity that allows to precisely identify the properties of pseudoknots that make them either hard or simple to align for the algorithm.

On the practical side, I focus on efficient implementations of the developed algorithms to make them accessible for biologists to support their research. To evaluate the usability in practice, I set up a pipeline that identifies conserved pseudoknots in potential non-coding RNAs and apply them to biological data of two different organismic species.

1.2 Overview

The remainder of this thesis is structured as follows. In Chapter 2 the relevant background is given. This includes both an introduction to biological aspects like RNA structure and pseudoknots as well as a description of the central formal concepts like alignments and their associated scoring schemes.

Chapter 3 develops a general perspective on dynamic programming (DP) with RNA structures with a focus on two tasks. First, it is shown how all existing DP based pseudoknot prediction algorithms can be explained in terms of a general framework. This is in particular important since an analogous framework for pseudoknot alignment will be developed later in the thesis. Second, the fundamental concepts that are common among DP based RNA alignment methods are identified. Those form the core of all alignment methods developed in the remaining chapters of the thesis.

Chapters 4 and 5 present two different approaches for the pairwise alignment of pseudoknots. The method presented in Chapter 4 is fixed parameter tractable and is able to align arbitrary pseudoknots. Despite the NP-hardness of the problem, this algorithm makes it tractable for many practical instances.

The second alignment method, presented in Chapter 5, is able to align only restricted classes of pseudoknots but with a polynomial time and space guarantee. The method is not restricted to one class of pseudoknots. Instead, a general scheme is developed that yields for each of the structure classes predicable by any of the seven pseudoknot prediction algorithms discussed in Chapter 3, a corresponding alignment algorithm. This scheme is obtained by lifting the general scheme for structure prediction developed in Chapter 3 to the alignment problem.

Finally, in Chapter 6 implementations of both alignment methods are presented and applied to real biological data. The time and space consumption of the approaches is measured and the faster approach is integrated into a pipeline that is able to detect conserved pseudoknots in potential non-coding RNAs. This pipeline is then applied to biological data of two different organisms.

1.3 Contributions

The main contributions of this thesis are two new methods for the alignment of RNA pseudoknot structures:

(a) a fixed-parameter tractable algorithm that is able to align arbitrary pseudoknot structures, and

(b) a general algorithm scheme that yield for various restricted classes of pseudoknots a polynomial alignment algorithm.

The fixed-parameter tractable alignment algorithm is the first exact alignment algorithm for arbitrary pseudoknots that is actually implemented and whose complexity makes it applicable in practice. In particular, the fixed parameter that

denotes the exponential factor in the complexity analysis, has a size of 1 for most practical instances.

The generality of the algorithm scheme for restricted pseudoknot classes is visible in the fact that it yields alignment algorithms for all classes of pseudoknots for which dynamic programming based structure prediction algorithms exist. For six of these seven classes, no alignment algorithms have been known so far. For the seventh, most general class, the new algorithm has a time and space complexity of $O(nm^6)$ and $O(nm^4)$, where n and m denote the length of the two aligned sequences. This is a significant improvement compared to the best previously known approach that requires $O(n^5m^5)$ time and $O(n^4m^4)$ space.

Merely as a side effect, the alignment algorithm scheme also shows how to embed all previously mentioned pseudoknot prediction algorithms in a common scheme. In summary, this results in a clear and new understanding of the available dynamic programming based methods for both pseudoknot prediction and alignment, in particular about their differences and similarities. Furthermore, this leads to new insights about the aspects of the algorithms that determine the essential trade off between their expressivity (i.e. class of covered pseudoknots) and their complexity.

1.4 Published Results

Both alignment approaches developed in this thesis have already been published in joint work with my coauthors Rolf Backofen and Sebastian Will. The presentation in this thesis differs significantly from the original papers. In particular, the current presentation highlights the similarities and differences of the two approaches by explaining them both in terms of the same syntax and based on the same core lemmata. Furthermore, the correctness proofs that are contained in the papers only in a condensed form, are given in full detail in this thesis.

Mathias Möhl, Sebastian Will, and Rolf Backofen. Fixed parameter tractable alignment of RNA structures including arbitrary pseudoknots. In*Proceedings of the 19th Annual Symposium on Combinatorial Pattern Matching (CPM 2008)*, LNCS, pages 69-81. Springer, 2008

Mathias Möhl, Sebastian Will, and Rolf Backofen. Lifting prediction to alignment of RNA pseudoknots. In *Proceedings of the 13th Annual International Conference on Computational Molecular Biology (RECOMB 2009)*, volume 5541 of LNBI, pages 285-301. Springer, 2009

2 Background

2.1 Biological Background

2.1.1 RNA and its Function in the Cell

RNA molecules that essentially occur in any living cell can be divided into two big classes: messenger RNA (mRNA) and non-protein-coding RNA (ncRNA). The biological function of messenger RNA is to convey the genetic information from DNA molecules located in the nucleus to the ribosomes where the mRNA is transcribed into proteins. The DNA in the nucleus is organized in chromosomes and each chromosome contains many genes. When a certain protein is expressed in the cell, the corresponding gene is first transcribed into a complementary nucleotide RNA strand. The resulting mRNA molecule is exported from the nucleus to the cytoplasm where by interaction with a ribosome its sequence is translated into a corresponding sequence of amino acids building a new protein. In eukaryotic organisms the mRNA also undergoes a certain processing before it is translated. Splicing mechanisms remove certain parts of the mRNA called introns and enable a single gene to encode several proteins.

For a long time is has been assumed that acting as mRNA is the predominant function of RNA. But there are also many kinds of non-protein-coding RNAs. Certain RNA molecules are, for example, involved in splicing and translation of mRNA. Furthermore some ncRNAs fulfill housekeeping functions, as telomerase RNA which is involved in DNA replication. Newer findings also revealed the existence of several riboregulators, i.e. ncRNAs that are involved in the regulation of gene expression [33, 34]. Another class of ncRNAs are ribozymes which are able to catalyze chemical reactions. The function of ribozymes is analogous to the function of enzymes with the only difference that the latter are proteins and not RNAs.

Apart from naturally occurring ncRNAs, also artificially designed RNA molecules are useful for both biological research and medical applications. RNA interference (RNAi) experiments, for example, allow a specific knock down of a gene with the help of artificially designed RNAs that bind to the respective mRNA and avoid it being translated into a protein.

Especially in complex organisms ncRNA seems to play an important role. This coincides with the fact that the amount of DNA that does not code for proteins (often considered as "junk") increases with the complexity of the organism: while in simple unicellular organisms 10-40% of the DNA is non-coding, in mammals this is the case for 98% of the DNA, as stated in [55]. In this article published in 2005, it is said that "it is now widely accepted that a significant fraction of the transcriptional

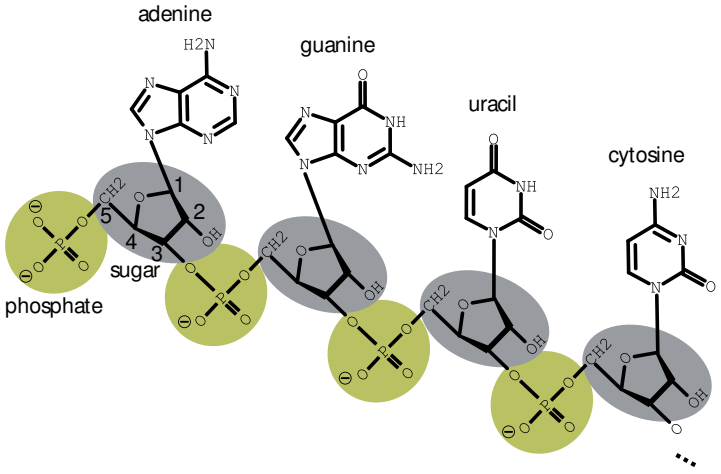


Figure 2.1: Chemical constitution of an RNA molecule.

output from the genome consists of untranslated or non-coding RNAs (ncRNA)". In 2007, also the first phase of the ENCODE project, a large scale project to identify functional elements in the genome, revealed that "the genome is pervasively transcribed, such that the majority of its bases can be found in primary transcripts, including non-protein-coding transcripts" [12]. These findings have renewed the interest in RNA analysis which has become a highly active field of research in both biology and bioinformatics.

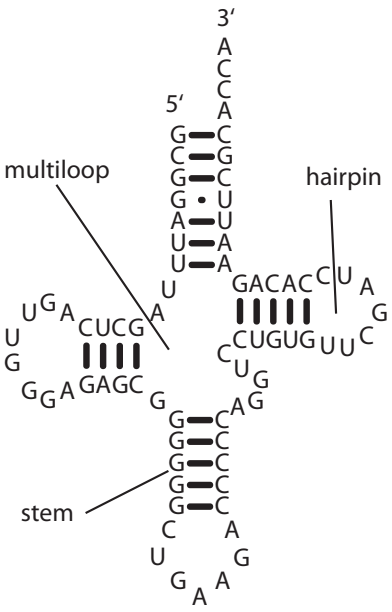
2.1.2 Chemical Constitution of the RNA Molecule

The chemical constitution of an RNA molecule is visualized in Figure 2.1. It consists of a chain of nucleotides whose backbone is built out of ribose sugar such that each nucleotide contains five carbon atoms denoted as C 1' to C 5'. A phosphate connects the C 5' of each nucleotide to the C 3' of the next nucleotide in the chain. The chains are hence asymmetric and usually considered in 5' to 3' direction. At each C 1', one out of the following four bases is attached: adenine (A), cytosine (C), guanine (G) and uracil (U). These bases are able to form so called Watson Crick base pairs [57], where A is able to pair with U and C is able to pair with G. In addition to the Watson Crick base pairs, sometimes also other pairs like the wobble base pair G-U occur. In contrast to DNA where base pairings lead to the formation of stable double helices, RNA is usually single stranded and pairings within this single strand lead to the formation of complex structures.

The RNA structure can be considered at different levels of abstraction that are visualized in Figure 2.2. The so called *primary structure* just considers the sequence of the bases in the chain and can be formally described as a string over the alphabet $\{A, C, G, U\}$. The primary structure of an RNA molecule is analogous to the primary structure of the DNA that it is transcribed from with the only difference that the

GCGGAUUUAGCUCAGUUGGGAGAGCGCCAGACUGAAGA
UCUGGAGGUCCUGUGUUCGAUCCACAGAAUUCGCACCA

(a) Primary structure.



(b) Secondary structure.

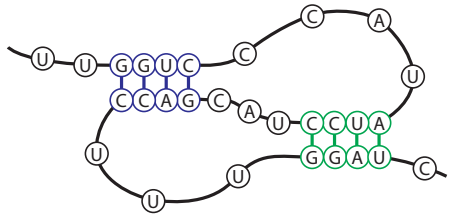
(c) tertiary structure (Reproduced with the kind permission of N.R. Voss).

Figure 2.2: Yeast phenylalanin tRNA ([28],PDB structure 1EHZ) visualized at different levels of abstraction.

DNA contains thymine (T) instead of uracil (U).

The *secondary structure* includes in addition to the pure sequence also the pairings among the bases. A secondary structure decomposes into different loops and stem structures. The stems are the regions of paired bases while the loops are formed by the unpaired parts. A loop that is adjacent to only one stem is called a hairpin loop and a loop that is adjacent to two stems is called an internal loop (or bulge, if it contains only one chain of unpaired bases). Loops with more that two adjacent stems are denoted as multi loops. Unpaired regions at the beginning or the end of the chain are called dangling ends.

The *tertiary structure* comprises the full 3D structure of the molecule. Since the tertiary structure is both difficult to determine experimentally and hard to predict, the amount of tertiary structure data available is far less than the amount of available primary and secondary structure information.



(a) pseudoknot drawn in the plane

(b) pseudoknot drawn linearly

```
..[[[[....((((...]]]]...)))).
UUGGUCCCAUAUCCUACGACCUUUGGAUC
```

(c) pseudoknot in dot bracket representation

Figure 2.3: Three different visualizations of a simple H-type pseudoknot. The red line indicates the backbone, black dots represent bases and the blue and green arcs represent pairings among bases.

Pseudoknots

Pseudoknots are motifs in the RNA secondary structure that do not simply decompose into loops and stems. The term pseudoknot was first introduced in 1978 by Studnicka et al. [54]. If we imagine the base pairs as arcs written on top of the sequence (Figure 2.3b), the characteristic feature of pseudoknots is that they contain crossing base pairs. Another convenient representation for secondary structures that we are going to use from time to time is the so called dot bracket representation shown in Figure 2.3c. Here, the sequence is augmented with a string that contains one dot for each unpaired base and one pair of brackets for each base pair. In this representation, the characteristic property of pseudoknots is that more than one type of bracket symbols is required to represent the structure.

Pseudoknots can be grouped into different classes according to their shape. Although no systematic scheme for this classification has been presented so far, some classes of interest have names that are well-established in the literature.

The simplest and most frequent form are so called H-type pseudoknots as shown in Figure 2.3 that just consist of two crossing stems. Another frequent type of pseudoknots are kissing hairpins. They occur if bases of two different hairpin loops pair to form an additional stem. An example for kissing hairpins is the Varkud Satelite (VS) RNA shown in Figure 2.4 that is found in certain natural isolates of *Neurospora*[43, 6].

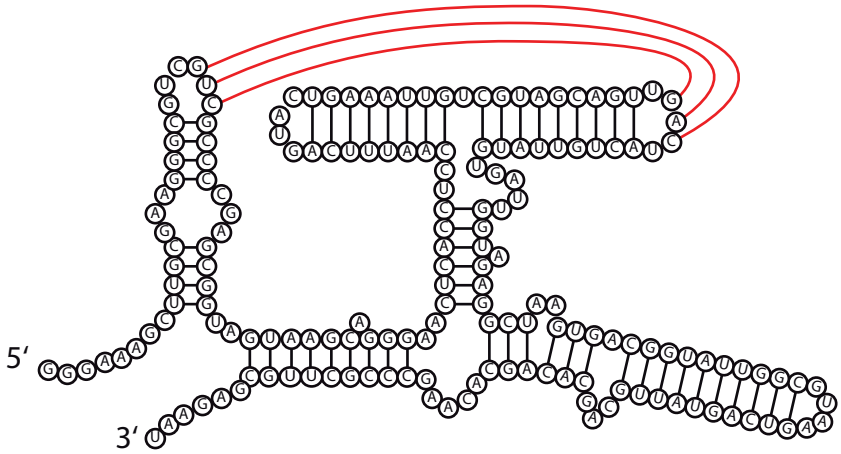


Figure 2.4: Varkud Satelite (VS) RNA with kissing hairpins. The bonds between the kissing hairpins are shown in red.

Pseudoknots are of interest because they occur frequently and, furthermore, because their presence is often required for the respective function of the RNA molecule. Hence, it is usually not adequate to ignore pseudoknot structures where they are present, although this is frequently done for simplification. The functional relevance of pseudoknots can be experimentally verified with mutagenesis experiments. Those experiments modify the RNA such that the pseudoknot cannot form anymore and check whether the function is still maintained. With these methods, for example, the pseudoknot of VS RNA in Figure 2.4 has been shown to be essential for the tertiary folding and self cleavage activity of the molecule [43, 6].

Besides the functional relevant pseudoknots that naturally occur in living cells, pseudoknots also occur in artificially designed RNAs. Such artificial sequences are of interest for both scientific and medical applications. They are obtained by in vitro evolution, a method that is used to artificially generate RNA molecules with some desired function. The method starts with a huge pool of random RNA sequences and applies a series of interleaved mutagenesis and filtering steps to obtain the sequences that bind to the desired target or perform another desired function. Using this technique, for example a self-alkylating ribozyme was found that contains a pseudoknot structure. Further experiments showed that this pseudoknot is responsible for the ability of the molecule to specifically bind to biotin [61, 60].

2.1.3 RNA Alignment and Comparative Methods

An alignment represents a comparison of structures and is used to identify similar as well as different parts among them. Alignments are a central part of the methods used in comparative genomics. Comparative genomics relies on the fact that functionally relevant parts are usually more conserved during evolution than others. This is due to evolutionary pressure that extinguishes mutations that lead to a loss

of function. Comparative methods are, for example, applied to DNA sequences to identify genes or regulatory regions that control gene expression. Typical applications combine alignments with other methods into a pipeline. Such a pipeline takes entire genomes or large candidate sets of sequence fragments as input and outputs a small set of candidates that satisfy certain properties (e.g. high evolutionary conservation or high similarity to some given pattern). The potential candidates identified with these methods are then usually further investigated in biological experiments.

While for DNA analysis simple sequence alignments are usually sufficient, in the case of RNA it is often desired to include the secondary structure in the comparison. Since the secondary structure of ncRNAs is usually related to their function, a comparison at this level yields more accurate results. Often the secondary structure is evolutionary much more conserved than the sequence that folds into that structure. This is possible because of so called compensatory base pair mutations, where bases of a base pair change but the structure is preserved. If, for example, a structure contains a G-C base pair and the G mutates into a U, then the base pair cannot form anymore since U cannot pair with C. But if the C simultaneously mutates into some A, then the base pair can form again as U-A and hence the structure is maintained although the underlying sequence has changed. The change of a U-A base pair to the wobble base pair U-G or vice versa is the most likely structure preserving mutation, since it only requires the mutation of one base of the pair.

The presence of compensatory mutations is relevant in two aspects: first, it indicates that the structure might be functionally relevant and second, that the hypothesized structure is correct. The latter aspect is in particular important, since the available methods to predict the structure have a low accuracy. Mathew*et al.* [32] compared 16 known RNA structures to the structures computed by state of the art MFE structure prediction and report an average accuracy of 76.4%. In the presence of pseudoknots, the prediction accuracy get even lower as the four RNase P variants show that where included in this study. These four RNAs had the largest percentage of pseudoknot base pairs (between 7.1% and 10.5%) among the considered RNAs and the accuracy of the prediction was only between 53.5% and 64.5%. Hence, in particular for pseudoknot structures comparative methods are required to get more precise predictions.

Variants of Alignment

There exist a lot of different alignment methods that vary with respect to several different aspects. The first aspect is the number of considered structures. While a pairwise alignment just compares two of them, a multiple alignment compares three or more structures. Since computing an entire multiple alignment in one step is usually prohibitively expensive, multiple alignments are often built in a progressive way, by combining pairwise alignments step by step. Commonly used program based on this idea are ClustalW [10] and T-Coffee [40].

A second aspect in which alignment methods differ is whether they compute a global or a local alignment. While global alignments compare the entire structures,

local alignment methods aim at identifying highly similar local parts. For the alignment of sequences, the classical algorithms for global and local alignments are the Needleman-Wunsch algorithm [39] and the Smith-Waterman algorithm [52], respectively. For RNA structures the LocARNA approach [41] is based on a more advanced notion of locality. Here, local motifs are not simple subsequences but local motifs with respect to the secondary structure.

Another criterion with respect to which existent alignment approaches differ is the level of structure which they consider. While for DNA usually plain sequence alignments are used, for RNA often the secondary structure level is considered in the comparison. Most approaches of secondary structure alignment are limited to pseudoknot-free inputs in order to guarantee a polynomial runtime [49, 59, 51, 27]. Since the presence of pseudoknots makes the problem quite complex, the only available approaches to align pseudoknots either have a high time and space complexity [20, 21] or do not guarantee to find an optimal solution [5].

Besides the approaches that work with given secondary structures there are also approaches that perform simultaneous alignment and folding [49, 25, 59, 41]. Those approaches get pure sequence data as input, but perform the comparison at the secondary structure level by predicting appropriate structures on the fly.

A last, more technical difference among alignment approaches is the choice of the scoring scheme according to which the optimal alignment is computed. The scoring schemes are usually based on some kind of edit distance measure that resembles evolutionary distance. The choice of the scoring scheme has important influence on both the quality of the resulting alignment (judged with respect to whether it is biologically meaningful or not) and the complexity of the alignment task.

In this thesis the focus is on pairwise, global alignments with given secondary structures that may contain pseudoknots. A general scoring scheme is employed throughout that subsumes many simpler schemes, but remains computationally tractable.

2.2 Formal Preliminaries

2.2.1 Formal Abstraction of RNA Structures

We formally represent RNA structures at the secondary structure level as arc-annotated sequences. The notion of arc-annotated sequences was first introduced by Evans [19]. Her PhD thesis gives a comprehensive overview over their properties, variants and related problems.

Definition 1 (arc-annotated sequence) An *arc-annotated sequence* is a pair (S, P), where S is a string over the set of bases $\{A, U, C, G\}$ and P is a set of arcs (l, r) with $1 \leq l < r \leq |S|$ representing bonds between bases, such that each base is adjacent to at most one arc. We denote the i-th symbol of S by $S[i]$. For an arc $p = (l, r)$, we denote its left end l and right end r by p^{L} and p^{R}, respectively. An arc $p \in P$ is *crossing* if there is an arc $p' \in P$ such that $p^{\mathrm{L}} < p'^{\mathrm{L}} < p^{\mathrm{R}} < p'^{\mathrm{R}}$ or

$p'^{\mathrm{L}} < p^{\mathrm{L}} < p'^{\mathrm{R}} < p^{\mathrm{R}}$. An arc-annotated sequence is *crossing* if it contains crossing arcs, otherwise it is *nested* or *non-crossing*. □

Pseudoknots correspond to crossing arc-annotated sequences. From now on, we consider two fixed arc-annotated sequences (S_a, P_a) and (S_b, P_b). Following [27], we define the functions ψ_k, $k \in \{a, b\}$ to denote whether position i of sequence k is adjacent to an arc and the function χ that checks whether the two sequences are equal at certain positions:

$$\psi_k(i) := \text{ if } \exists j : (i,j) \in P_k \text{ or } (j,i) \in P_k \text{ then 1 else 0} \quad (\text{for } k = a, b)$$
$$\chi(i,j) := \text{ if } S_a[i] \neq S_b[j] \text{ then 1 else 0}$$

2.2.2 Alignment and Scoring Schemes

An alignment can be considered as an order-preserving, partial mapping from the positions of one arc-annotated sequence to a second one. For technical reasons we extend the partial mapping to a total one by introducing so called *gap edges*.

Definition 2 (alignment) An *alignment* A of two arc-annotated sequences (S_a, P_a) and (S_b, P_b) is a set $A = A_M \cup A_G$, where $A_M \subseteq [1..|S_a|] \times [1..|S_b|]$ is a set of *match edges* such that for all $(i,j),(i',j') \in A_M$ holds 1.) $i > i'$ implies $j > j'$ and 2.) $i = i'$ if and only if $j = j'$ and A_G is the set of *gap edges* $\{\, (x,-) \,|x \in [1..|S_a|] \wedge \nexists y.(x,y) \in A_M \,\} \cup \{\, (-,y) \,|y \in [1..|S_b|] \wedge \nexists x.(x,y) \in A_M \,\}$ where $-$ is a dedicated gap symbol. A base that is adjacent to a gap edge is called *aligned to a gap*. Two bases $S_a[i]$, $S_b[j]$ are matched by A if $(i,j) \in A$ and two arcs $p_a \in P_a$, $p_b \in P_b$ are matched if $(p_a^{\mathrm{L}}, p_b^{\mathrm{L}}) \in A$ and $(p_a^{\mathrm{R}}, p_b^{\mathrm{R}}) \in A$.[1] □

An alignment is usually visualized as shown in Figure 2.5 by putting the two sequences with inserted gaps on top of each other such that matched bases are located on top of each other. The optimal alignment is usually formalized as the alignment of minimum cost with respect to some given cost function. The complexity of the task to find the minimum cost alignment heavily depends on this cost function as we will see in Section 4.1. Cost functions are typically formulated as the sum of the costs of certain local fragments. Since the costs of these fragments are independent, dynamic programming algorithms are able to split the problem to find the optimal alignment, into independent sub problems. In turn, the problem to find the optimal alignment is the more simple the smaller and less complex these local fragments are.

In our scoring scheme, we consider three kinds of local fragments. We consider a cost for each base $S_a[i]$ or $S_b[j]$ that is aligned to a gap. This cost is denoted as $\mathrm{gap}_a(i)$ or $\mathrm{gap}_b(j)$, respectively. Then we consider a cost $\mathrm{arcmatch}(p_a, p_b)$ for each pair (p_a, p_b) of matched arcs and also a cost $\mathrm{basematch}(i,j)$ that represents the cost to align a base $S_a[i]$ to a base $S_b[j]$ given that this base match is not part of an arc match (i.e. given that for all arc matches (p_a, p_b) holds $(i,j) \neq (p_a^{\mathrm{L}}, p_b^{\mathrm{L}})$ and $(i,j) \neq (p_a^{\mathrm{R}}, p_b^{\mathrm{R}})$).

[1] Note that by this definition base matches and arc matches do not exclude each other, i.e. if two arcs are matched, their respective left and right ends also form base matches.

Definition 3 (cost of an alignment) The cost of an alignment A is defined recursively as $\text{cost}(A)$ with

$$\begin{aligned}
\text{cost}(\{(i,-)\} \uplus A') &= \text{gap}_a(i) + \text{cost}(A'), \\
\text{cost}(\{(-,j)\} \uplus A') &= \text{gap}_b(j) + \text{cost}(A') \\
\text{cost}(\{(l_a,l_b),(r_a,r_b)\} \uplus A') &= \text{arcmatch}((l_a,r_a),(l_b,r_b)) + \text{cost}(A') \\
&\qquad \text{if } (l_a,r_a) \in P_a, (l_b,r_b) \in P_b \\
\text{cost}(\{(i,j)\} \uplus A') &= \text{basematch}(i,j) + \text{cost}(A') \\
&\qquad \text{if third case is not applicable.}
\end{aligned}$$

□

This scoring scheme is chosen such that it is as general as possible while still being valid for the alignment algorithms developed in this thesis. Its central property is that all costs except the cost for an arc match depend only on a single position of each sequence. In that sense an arc match is the only non-locality that the algorithms have to cope with. Note in particular, that if an arc is matched to something else that an arc (i.e. if the arc is broken), the costs of the alignment at the two ends of the arc are mutually independent.

General Edit Distance The scoring scheme presented above subsumes the *general edit distance* proposed by Jiang *et al.* [27] for what they call *reasonable scoring schemes*. Since the general edit distance is biologically motivated, it is an interesting instance for practical applications.

The general edit distance is based on the operations illustrated in Figure 2.5. Base operations (mismatch and insertion/deletion) work solely on positions that are not incident to an arc. *Base mismatch* replaces a base with another base and has associated cost w_m. A *base insertion/deletion* removes or adds one base and costs w_d. The second class consists of operations that involve at least one position that is incident to an arc. An *arc mismatch* replaces one or both of the bases incident to an arc. It costs $\frac{w_{am}}{2}$ if one base is replaced or w_{am} if both are replaced. An *arc breaking* removes one arc and leaves the incident bases unchanged. The associated cost is w_b. *Arc removing* deletes one arc and both incident bases and costs w_r. Finally, *arc altering* removes one of the two bases that are incident to an arc and costs w_a.

For what Jiang *et al.* call *reasonable scoring schemes*, the cost of arc altering is restricted to $w_a = \frac{w_r}{2} + \frac{w_b}{2}$. This constraint is required to satisfy the locality properties of our scoring scheme and without the constraint the problem becomes NP-hard (see Section 4.1). To represent the edit distance in our general scoring scheme, we instantiate our cost functions as follows.

$$\begin{aligned}
\text{gap}_k(i) &:= w_d + \psi_a(i)(\frac{w_r}{2} - w_d) \quad (\text{for } k = a, b) \\
\text{basematch}(i,j) &:= +\chi(i,j)w_m + (\psi_a(i) + \psi_b(j))\frac{w_b}{2} \\
\text{arcmatch}(p_a,p_b) &:= (\chi(p_a^{\text{L}},p_b^{\text{L}}) + \chi(p_a^{\text{R}},p_b^{\text{R}}))\frac{w_{am}}{2}
\end{aligned}$$

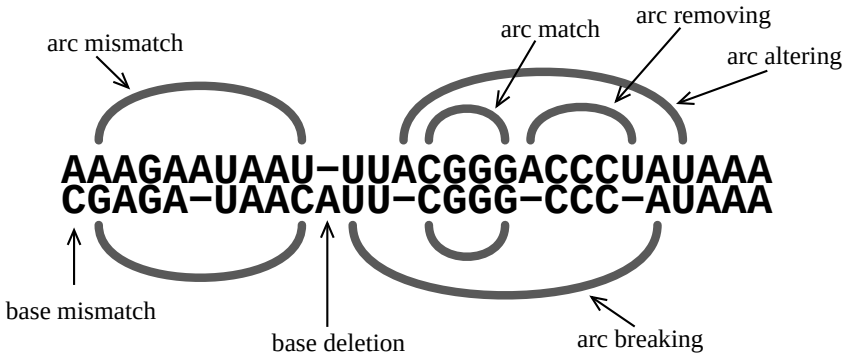


Figure 2.5: Edit operations of the general edit distance according to Jiang *et al.* [27]

A gap corresponds to a base deletion operation if the respective base is not adjacent to an arc. Otherwise it is caused by an arc removing or an arc altering operation with cost $\frac{w_r}{2}$ in both cases due to the constraint $w_a = \frac{w_r}{2} + \frac{w_b}{2}$. Similarly, in a base match possibly adjacent arcs need to be broken which causes cost $\frac{w_b}{2}$ for each arc, no matter whether this is done in an arc breaking or an arc altering operation.

Other Scoring Schemes Besides the general edit distance also tree edit distance and tree alignment are frequently considered. Both require to interpret the arc-annotated sequences as trees which is straight forward for nested structures. Since it is unclear how to generalize this concept to pseudoknots they are not considered in the following.

Other scoring schemes, in particular studied by Evans, are the longest arc-preserving common sub sequence (LAPCS) [19, 20] and the maximum common ordered substructure [21]. The alignment problem for LAPCS is NP-complete even for nested structures which makes it intractable for most practical purposes. On the other hand, the maximum common ordered substructure just maximizes the number of arc matches (including matched arcs where the adjacent bases do not match) and hence corresponds to the instance of the general edit distance where $w_m = 0$, $w_d = 0$, $w_{am} = 0$, $w_b = 1$, $w_r = 1$, and $w_a = \frac{w_r}{2} + \frac{w_b}{2}$.

Blin *et al.* [7] compare various scoring schemes in the so called alignment hierarchy. As will be described in Sect. 4.1, among all edit problems considered in this study, the general edit distance with the constraint on arc altering is the most general scheme for which at least the alignment problem for pseudoknot-free input structures is solvable in polynomial time. All edit problems with independent arc-altering and arc-removing operations considered there where shown to be NP-hard.

3 A General Approach to Dynamic Programming on RNA structures

Dynamic programming algorithms make use of the recursive structure of the objects they are working on. This chapter analyzes the recursive structure of RNA and makes three important contributions.

First, we show how RNA structures can be recursively described with the help of parse trees (Section 3.1). While the correspondence between pseudoknot-free RNA structures and trees is well known and used in many applications, for pseudoknotted structures no such concept existed so far.

As a second contribution we confirm the usefulness of the parse tree concept by analyzing various algorithms for RNA pseudoknot prediction (Section 3.2). It turns out that all these algorithms can be understood as (implicitly) constructing parse trees and that both the complexity of the algorithms and the class of pseudoknots that they are able to predict are a direct consequence of the class of parse trees that the respective algorithm considers.

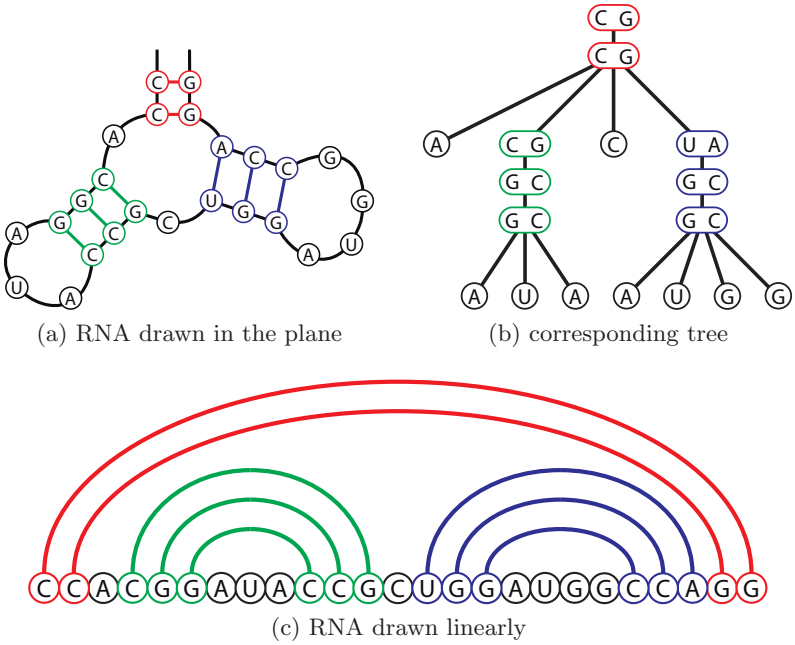
After having described all known dynamic programming based structure prediction algorithms in terms of a general framework based on parse trees, the last important contribution of this chapter is the development of the core of a similar framework for RNA alignment (Section 3.3). This framework forms the basement for all algorithms developed in the later chapters of this thesis.

3.1 A recursive view on RNA structures

3.1.1 Nested RNA Structures as Trees

Nested RNA structures have a quite natural correspondence to trees. Several different variants of how to associate an RNA structure with a tree have been proposed [50, 2] and employed to use tree alignment and tree edit distance algorithms for RNAs. The different encodings all rely on a tree structure induced by the set of arcs P of an arc annotated sequence and only differ in the ways how this tree structure is generalized to incorporate the sequence information.

If an arc-annotated sequence (S, P) contains no pseudoknots, then for each pair of distinct arcs p, p' either one arc precedes the other one ($p^{\mathrm{L}} < p^{\mathrm{R}} < p'^{\mathrm{L}} < p'^{\mathrm{R}}$ or $p'^{\mathrm{L}} < p'^{\mathrm{R}} < p^{\mathrm{L}} < p^{\mathrm{R}}$) or one of the two subsumes the other one. We define that an arc p subsumes and arc p', short $p \succ p'$, if $p^{\mathrm{L}} < p'^{\mathrm{L}} < p'^{\mathrm{R}} < p^{\mathrm{R}}$. Obviously the relation $\succ$ is acyclic and further more if $p' \succ p$ and $p'' \succ p$ then either $p' \succ p''$ or $p'' \succ p'$. Hence, the transitive reduction of $\succ$ is unique and forms a tree. In order to



(a) RNA drawn in the plane (b) corresponding tree

(c) RNA drawn linearly

Figure 3.1: A nested RNA structure and its correspondence to a tree. The stems are highlighted in different colors to show the correspondence.

also represent positions of the sequence that are not adjacent to an arc, this basic tree can be extended with a set of leaves where each leave corresponds to one of these unpaired positions. Technically this corresponds to adding one dummy arc (i, i) to P for each unpaired position i. An example for such a tree and the corresponding RNA structure is given in Figure 3.1.

Since such a tree is recursively composed out of subtrees, it represents a way to recursively compose the corresponding RNA structure. Each subtree with some arc p as root node corresponds to the subsequence containing the positions $[p^{\mathrm{L}}, p^{\mathrm{R}}]$. Hence the entire structure is recursively composed by concatenating contiguous fragments.

3.1.2 Pseudoknot Structures as Parse Trees

Pseudoknotted RNA structures have no direct correspondence to trees as nested structures have. The reason for this is that if two arcs p and p' are crossing, neither of them subsumes the other, but any arc p'' within the region in which p and p' overlap, is subsumed by both p and p'. Hence p'' would need to contained in both the subtrees rooted at p and p' although those must be disjoint.

In order to find a useful generalization for pseudoknot structures it is important to identify the relevant properties that the correspondence between the arc annotated sequences and the trees has. The correspondence for nested structures has two important properties. First, each subtree corresponds to a contiguous fragment of the sequence and second, each of these fragments is arc complete in the sense

that no arc has one end within the fragment and one end outside. The latter property is important since it ensures a kind of independence that is necessary for the dynamic programming algorithms. Compared to that, the restriction to consider only contiguous fragments helps to reduce complexity but is not as vital. Since both properties cannot be maintained simultaneously for pseudoknot structures, the framework presented now shows how to recursively decompose pseudoknot structures into arc complete fragments that are not necessarily contiguous but may contain gaps.

The decision to allow gaps in the fragments in order to maintain arc completeness is also supported by existing RNA structure prediction algorithms. As will be shown in the next section, all dynamic programming based algorithms for RNA structure prediction recursively construct their structures out of gapped, arc complete fragments. We now give formal definitions for the relevant concepts.

Definition 4 (fragment) A *fragment* F of an arc annotated sequence (S, P) is a k-tuple of *intervals* $([l_1, r_1], \ldots, [l_k, r_k])$ with $1 \leq l_1 \leq r_1+1 \leq \cdots \leq l_k \leq r_k+1 \leq |S|$. The ranges between the intervals, i.e $[r_1 + 1, l_2 - 1], \ldots, [r_{k-1} + 1, l_k - 1]$, are called *gaps of* F. We call k the *degree* of F and $l_1, r_1, \ldots, l_k, r_k$ its *boundaries*. The *set of positions covered by* F (denoted with $\hat{F}$), is defined as the union of the intervals contained in F (where we technically understand an interval $[i, j]$ as a representation of the set $\{i, \ldots, j\}$). The i-th interval $[l_i, r_i]$ of F is denoted with $F[i]$ and with $F[i]^{\mathrm{L}}$ and $F[i]^{\mathrm{R}}$ we denote its *left and right boundary* l_i *and* r_i, respectively. □

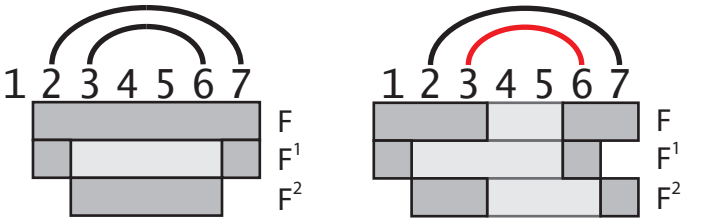
Note that this definition allows *empty intervals* $[i, i - 1]$. For better readability, we abbreviate intervals of the form $[i, i]$ as $[i]$. As mentioned before, the RNA structure should be recursively decomposed into fragments that are arc complete. Furthermore, the smallest fragments that are not decomposed any further should be atomic.

Definition 5 (properties of fragments) Let F be some fragment of an arc annotated sequence (S, P). An arc $p = (l, r)$ is called *open in* F, if and only if $l \in \hat{F} \not\Leftrightarrow r \in \hat{F}$. The fragment F is called *arc-complete* with respect to a subset $P' \subseteq P$ if and only if there does not exist an arc $p \in P$ that is open in F. If F is arc complete with respect to P we just say it is arc complete. F is called *atomic* if F covers either exactly the two ends of an arc of P or a single position not adjacent to an arc. □

The notion of decomposing a fragment into smaller fragments is captured in the concept of a split.

Definition 6 (split) Let F, F^1 and F^2 be fragments of the same sequence. The pair (F^1, F^2) is a *split* of F iff $\hat{F} = \hat{F^1} \uplus \hat{F^2}$.[1] We call F^1 and F^2 the *children* and F the *parent of the split*. The split is called *arc-preserving*, if F, F^1, and F^2 are arc complete. □

[1] For simplicity, we introduce only binary splits. However, the introduced concepts can be simply extended to n-ary splits.



(a) arc preserving split (b) split that is not arc preserving

Figure 3.2: Two examples for splits of fragments. The intervals of the fragments are drawn in dark and the gaps in light gray. The split in (a) is arc preserving and has basic split type 121 and constrained types $1'21'$, $1'21$, and $121'$. The split in (b) has basic type $12G12$, strongest constrained type $\downarrow 1'2G1'2'\downarrow$, and is not arc preserving, since the arc $(3, 6)$ is open in both F^1 and F^2.

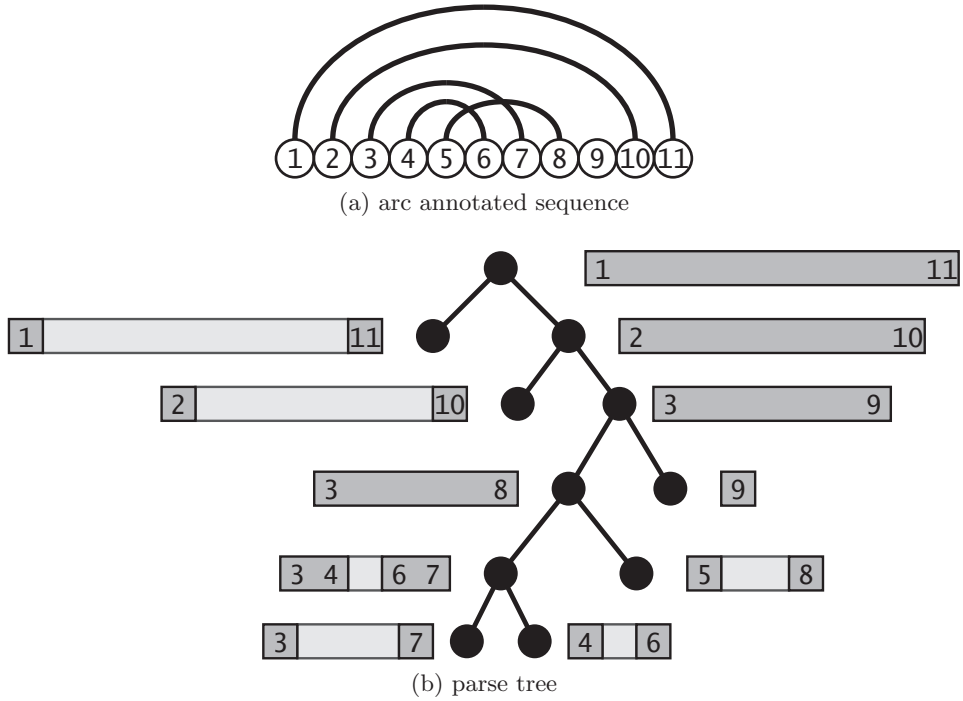
With splits we associate two kinds of types: basic types and constrained types. The basic type of a split captures the general form of the split and a constraint type represents in addition several constraints that the split satisfies.

Definition 7 (basic split type) The *basic type* of a split (F^1, F^2) of a fragment F is defined by the following construction. The interval $[\min(\hat{F}), \max(\hat{F})]$ decomposes into the intervals of F^1, the intervals of F^2 and gaps of F. If we order these from left to right and replace the intervals of F^1 by 1, the ones of F^2 by 2 and the remaining ones by G (for gap), we obtain a string T over $\{1, 2, G\}$ that we call the *basic type* of the split. □

Definition 8 (constrained split type) A *constrained type* of a split (F^1, F^2) of a fragment F is its basic type together with some annotated constraints of the following form.

- A length constraint on some interval is marked with a $'$ on the respective symbol 1 or 2 and indicates that the length of the interval is at most one.
- A maximality constraint is marked with a $\downarrow$ at the beginning and the end of the type and requires that the first and last boundary of F coincide with the beginning and the end of the sequence. □

Figure 3.2 gives two examples for splits and their associated basic and constrained types. While each split has exactly one basic type, it may have several constrained types. Constraint types will later be relevant to describe important optimizations of the considered algorithms that reduce their complexity significantly. With the general notion of a split, there are many possibilities to recursively decompose a pseudoknot structure into arc complete fragments. Each of them can be described as a parse tree.



(a) arc annotated sequence

(b) parse tree

Figure 3.3: A parse tree for some arc annotated sequence.

Definition 9 (parse tree) A *parse tree of a sequence* (S, P) is a binary tree where each node is an arc-complete fragment of (S, P) such that (a) the root is $([1, |S|])$, (b) each inner node is a fragment F and has two children F_1 and F_2, such that (F_1, F_2) is an arc-preserving split of F, (c) each leaf is an atomic fragment. □

A visualization of a parse tree is given in Figure 3.3. Each pseudoknot structure can be associated with a parse tree in a similar way as nested structures are usually associated with trees. The following important differences remain however.

- The correspondence between pseudoknot structures and parse trees is not unique, i.e. usually there exist many possible parse trees for a structure.
- The fragments associated with a parse tree contain gaps if the structure contains pseudoknots.

3.2 Classification of Pseudoknot Prediction Algorithms

In this section the notions of parse trees and splits introduced in the previous section are used to classify various existing pseudoknot prediction algorithms. It turns out that they all fit into a general scheme that is able to relate the class of structures that the respective algorithm is able to predict to the resulting time and space complexity.

Apart from the fact that this is an interesting result on its own, the gained insights are later used in Chapter 5 to develop an analogous scheme for pseudoknot alignment that yields one tailored alignment algorithm for each of the prediction algorithms.

3.2.1 The Recursive Structure of RNA Structure Prediction Algorithms

In contrast to pseudoknot alignment, for pseudoknot structure prediction there exist already various approaches that are based on dynamic programming [31, 46, 56, 1, 16, 17, 9, 44]. All those algorithms are dynamic programming based and compose the optimal structure for some given sequence recursively out of optimal structures of fragments. We analyze these algorithms now at a coarse grained level of abstraction that discards aspects like the scoring scheme and only focuses on the kinds of splits that are used to compose the optimal structure out of substructures.

At this level, a structure prediction algorithm can be characterized by a set of split types $\mathcal{T}$ that may contain both basic and constrained types. When computing the optimal structure for some fragment F, the algorithm iterates over all the instances of all split types in $\mathcal{T}$ for which F is the parent fragment and chooses among all those splits the optimum.

If we denote the optimal score for a structure of a fragment F with $\mathrm{OPT}(F)$ and assume that this score is minimized, the recursion of all considered algorithms follows the following scheme.

$$\mathrm{OPT}(F) = \min_{T \in \mathcal{T}} \left\{ \min_{T\text{-split } (F^1,F^2) \text{ of } F} \left\{ \mathrm{OPT}(F^1) + \mathrm{OPT}(F^2) \right\} \right\}. \tag{3.1}$$

Due to their scoring scheme most algorithms actually consider optimal scores for different kinds of structures, as for example the optimum among all loop structures or among all pseudoknot structures. However, this does not affect the general structure of the recursion. The only difference is that those algorithms do not compute one value $\mathrm{OPT}(F)$ for a fragment F, but a constant number of values $\mathrm{OPT}_1(F)$, ..., $\mathrm{OPT}_k(F)$, where each of them represents the optimum among only a certain kind of structures.

We illustrate the scheme with a simple example. The well-known Zuker algorithm [63] predicts pseudoknot-free structures and computes two matrices W and V where W represents the optimum among all structures and V the optimum among only the structures where the first and last position are paired. The recursion for W is

$$W[i,j] = \min \begin{cases} W[i+1,j] & \text{(3.2a)} \\ W[i,j-1] & \text{(3.2b)} \\ \min\limits_{i<k<j-1} \{W[i,k] + W[k+1,j]\} & \text{(3.2c)} \\ V[i,j] & \text{(3.2d)} \end{cases}$$

At our coarse grained level of abstraction, the recursion corresponds to the instance of Equation (3.1) with $\mathcal{T} = \{1'2, 12', 12, 1\}$. The type $1'2$, for example, corresponds

to case (3.2a), since this case splits the structure into the single position i (in this case with a contribution of 0 to the score) and the remaining fragment $[i+1, j]$. Note that for fixed $F = ([i, j])$, only the third type, 12 has more than one instance. Therefore case (3.2c) is the only one that has to minimize over a set of values. The final set of split types for the Zuker algorithm would be the union of the sets of split types for both matrices W and V.

For pseudoknot structure prediction algorithms the set of split types contains split types with gaps. This is necessary to construct a pseudoknot out of two crossing stems. A simple pseudoknot can be constructed, for example, with the split type 1212 to compose the two stems and the type $1'2G21'$ which can recursively construct a stem by adding an additional outermost arc. Note that a single arc is represented in the split type by two intervals with a length constraint.

3.2.2 The Relation between Parse Trees, Classes of Pseudoknots and Algorithmic Complexity

The abstraction of a structure prediction algorithm via recursion 3.1 and an associated set of split types $\mathcal{T}$ suffices to analyze both the complexity of the algorithm and the class of structures that it is able to predict. A structure can be predicted by some algorithm if the structure can be recursively composed out of fragments according to the recursive structure of the algorithm. In other words, the class of structures that an algorithm is able to predict equals the class of structures for which parse trees exist that contain only splits of types contained in $\mathcal{T}$. This insight allows to easily compare the classes of structures predicted by different structure prediction algorithms. The more and the more complex split types are contained in $\mathcal{T}$, the larger is the class of predicted structures. In particular, if for two algorithms the respective sets $\mathcal{T}$ are in a subset relation, this subset relation also holds for the respective classes of structures. A comparison of the known pseudoknot prediction algorithms with respect to such inclusion relations has already been done by Condon *et al.* [11]. The characterization of the algorithms in terms of the split types confirms those results and explains them in a new, intuitive way.

From that perspective, designing a good structure prediction algorithm consists to some extend of finding a set of split types with which one can on the one hand parse as many pseudoknots as possible and that is on the other hand as small and simple as possible. The latter is important since it influences the complexity of the algorithm.

A time and space complexity analysis of the prediction algorithms can be done based directly on Equation (3.1). If the equation is applied recursively, values OPT(F) need to be computed for all fragments F that are a parent instance of some split T of a split type contained in $\mathcal{T}$. We denote the number of parent instances and child instances of a split T for a structure of length n with $\#_P^n(T)$, and

$\#_C^n(T)$, respectively:

$$\#_P^n(T) = \left| \left\{ F \,\middle|\, \exists (F^1, F^2) \text{ that is a T-split of } F \text{ and } \hat{F} \subseteq [1,n] \right\} \right|$$
$$\#_C^n(T) = \left| \left\{ (F^1, F^2) \,\middle|\, (F^1, F^2) \text{ is a T-split of some } F \text{ and } \hat{F} \subseteq [1,n] \right\} \right|$$

If $|\mathcal{T}|$ is assumed to be constant the number of values F that need to be computed is

$$\left(\sum_{T \in \mathcal{T}} \#_P^n(T) \right) \in O(\max_{T \in \mathcal{T}} \#_P^n(T))$$

and the total time required for the computation of all these values is

$$\left(\sum_{T \in \mathcal{T}} \#_C^n(T) \right) \in O(\max_{T \in \mathcal{T}} \#_C^n(T))$$

since the computation of each parent of a split minimizes over all possible children. In other words, the time and space complexity of a structure prediction algorithm is directly implied by the number of parent and children instances, respectively, of its most complex split type.We now derive further bounds for $\#_C^n(T)$ and $\#_P^n(T)$ for simple and constrained types.

Lemma 1 (number of instances of basic types) *For a sequence with length n and a basic split type T, let the degree of the parent and the two children be k_p, k_1, and k_2, respectively. Then $\#_C^n(T) \in O(n^{k_p+k_1+k_2})$ and $\#_P^n(T) \in O(n^{2k_p})$.* □

Proof Each instance of a fragment of degree k is uniquely determined by its $2k$ boundaries and each boundary has one of the values $1 \ldots n$. Hence $\#_P^n(T) \in O(n^{2k_p})$.

Each split is determined by the $2(k_p + k_1 + k_2)$ boundaries of the parent and the two children. Every two of them depend on each other: each parent boundary must coincide with some child boundary and from the remaining boundaries of the children, always two are directly adjacent. Hence, $k_p + k_1 + k_2$ values can be chosen to determine each instance. Hence $\#_C^n(T) \in O(n^{k_p+k_1+k_2})$. ■

A maximality constraint reduces the number of instances of a type by a quadratic factor since it fixes the first and last boundary. A length constraint reduces the number of instances by a linear factor since it creates a dependency among the two boundaries of the fragment. Hence the bounds on $\#_C^n(T)$ and $\#_P^n(T)$ can be refined for constraint types.

Lemma 2 (number of instances for constrained types) *For a sequence with length n and some constraint type T, let the degree of the parent and the two children be k_p, k_1 and k_2, respectively and let c_l be the number of length constraints on T and $c_m = 2$ if T has a maximality constraint and 0 otherwise. Then $\#_C^n(T) \in O(n^{k_p+k_1+k_2-c_l-c_m})$ and $\#_P^n(T) \in O(n^{2k_p-c_m})$.* □

PROOF The argumentation is the same as for Lemma 1. In addition note that a maximality constraint fixes the first and last boundary of the fragment. Since the first and last boundary are always also boundaries of the parent, this reduces both $\#_C^n(T)$ and $\#_P^n(T)$ by a quadratic factor. For each length constraint, if the left boundary of the interval is at position i, the right one can only be instantiated with one of the two values $i-1$ and i since the interval must have length 0 or 1. Hence $\#_C^n(T) \in O(n^{k_p+k_1+k_2-c_l-c_m}2^{c_l}) = O(n^{k_p+k_1+k_2-c_l-c_m})$ if we assume c_l to be a constant. ■

As suggested by Lemma 2 the time and space requirements of an algorithm are lower if the types in $\mathcal{T}$ contain constraints. Most pseudoknot prediction algorithms take advantage of this fact. A further possibility used by many of the algorithms to reduce the space complexity is based on certain invariants among the types T contained in $\mathcal{T}$.

Improving Space Complexity with Invariants The recursion scheme in Equation (3.1) does not say anything about the order in which the values are computed. Obviously the order must be such that the values for smaller fragments must be computed before the values for larger ones that contain them, but there is still a lot of freedom in the choice of an evaluation order. In particular, if the types contained in $\mathcal{T}$ satisfy certain invariants, the evaluation order can be chosen such that not all entries must be kept in memory at the same time. To make this possible, the fragments must be grouped into subsets that do not depend on each other recursively.

Definition 10 (grouped fragments) A *grouping of a set of fragments* $\mathcal{F}$ into k groups is a collection of sets $\mathcal{S}_1$,...,$\mathcal{S}_k$, $\mathcal{G}_1$,...,$\mathcal{G}_k$ such that $\mathcal{F} = (\mathcal{S}_1 \uplus \cdots \uplus \mathcal{S}_k) \uplus (\mathcal{G}_1 \cup \cdots \cup \mathcal{G}_k)$. We call the elements in any $\mathcal{S}_i$ simple fragments and the ones in any $\mathcal{G}_i$ grouped fragments. Note that the groups $\mathcal{G}_i$ may overlap, which corresponds to recomputing a value several times in order to save space. Let $\mathcal{F}$ now be the set of fragments that occur as child and parent fragments in the instances of a set of split types $\mathcal{T}$. Then we call $\mathcal{T}$ *invariant with respect to the grouping* if for all $T \in \mathcal{T}$, all T-splits (F^1, F^2) of some F and all i holds

$$F \in \mathcal{G}_i \cup \mathcal{S}_i \quad \Rightarrow \quad F^1 \in \mathcal{G}_i \cup \mathcal{S}_1 \cup \cdots \cup \mathcal{S}_i \wedge F^2 \in \mathcal{G}_i \cup \mathcal{S}_1 \cup \cdots \cup \mathcal{S}_i \tag{3.3}$$

$$|\{\, \mathcal{G}_i \,|F \in \mathcal{G}_i \,\}| \text{ is bounded by some constant} \tag{3.4}$$

□

The intuition of a grouping is that first the elements of $\mathcal{S}_1 \cup \mathcal{G}_1$ are computed, then the elements of $\mathcal{S}_2 \cup \mathcal{G}_2$ and so on. Furthermore, the elements of any $\mathcal{S}_i$ are maintained in memory forever once they are computed while the memory for a group $\mathcal{G}_i$ is freed as soon as the values of $\mathcal{G}_{i+1}$ are computed such that only one $\mathcal{G}_i$ is kept in memory at the same time. This is possible, since Requirement (3.3) ensures that elements of any group may only rely on grouped fragments of the same group and on any previously computed simple fragments. If the groups are not disjoint, the respective values are recomputed for each of the groups, but by Requirement (3.4)

each value is recomputed only a constant number of times and hence does not affect the asymptotic time complexity.

As an example, consider $\mathcal{T} = \{121, 1G21, 21G1\}$. For all split types in $\mathcal{T}$ the second child is always simple and the first child has always its last boundary in common with the parent fragment. In other words, during the recursive computation of a fragment, among the gapped fragments that are involved, the last boundary remains invariant. Therefore a grouping according to this last boundary satisfies the Requirement (3.3):

$$\mathcal{S}_i := \left\{ F \,\middle|\, F \text{ has degree 1 and } F[1]^{\mathrm{R}} = i \right\} \tag{3.5}$$

$$\mathcal{G}_i := \left\{ F \,\middle|\, F \text{ has degree 2 and } F[2]^{\mathrm{R}} = i \right\} \tag{3.6}$$

In this example the groups are disjoint and hence no value is computed more than once. There exist only $O(n^2)$ simple fragments and of the in total $O(n^4)$ fragments that contain at most one gap, each group only contains $O(n^3)$ entries. Hence an evaluation order according to the grouping would reduce the space requirements by a linear factor. The following lemma makes this optimization idea more precise.

Lemma 3 (space complexity for grouped fragments) *Let $\mathcal{T}$ be invariant for some grouping $\mathcal{S}_1 \cdots \mathcal{S}_k \cup \mathcal{G}_1 \cdots \mathcal{G}_k$. Furthermore, let f be a function such that*

$$\left| \left\{ F \in \mathcal{S}_1 \cup \cdots \cup \mathcal{S}_k \,\middle|\, \hat{F} \subseteq [1,n] \right\} \right| \in O(f(n)) \tag{3.7}$$

and for all i

$$\left| \left\{ F \in \mathcal{G}_i \,\middle|\, \hat{F} \subseteq [1,n] \right\} \right| \in O(f(n)) \tag{3.8}$$

Then the value $\mathrm{OPT}(F)$ for some fragment F of size n can be computed according to Recursion (3.1) in $O(f(n))$ space without the need to compute any value more than a constant number of times. □

Proof The computation is done in k stages. In each stage i the values for the elements of $\mathcal{S}_i \cup \mathcal{G}_i$ are computed and then the memory for $\mathcal{G}_i$ is freed before stage $i+1$ is entered. By that, each element of each group is computed exactly once and hence by Requirement (3.4) each value is only computed a constant number of times. Since at each point in time at most values for one group $\mathcal{G}_i$ are kept in memory, the space requirement of $O(f(n))$ follows directly from (3.7) and (3.8).

It remains to show that for each stage i there exists an evaluation order such that for each value that is computed the values that it depends on are already computed and not yet erased again from memory. First note that by the nature of a split, each fragment F only depends on fragments F' for which $|\hat{F}'| < |\hat{F}|$. Hence there are no cyclic dependencies which implies that there exists an evaluation order that at least satisfies all dependencies among fragments computed within the same stage. By Requirement (3.3) the only dependencies on values that are computed in another stage are dependencies on simple values computed in an earlier stage. Since the simple values are maintained forever in memory once they are computed, those dependencies also do not conflict with the evaluation order. ■

Table 3.1: Pseudoknot classes and complexity of their associated prediction algorithms. For A&U the first value is for simple and the second for simple recursive structures.

class		R&E	A&U	L&P	D&P	CCJ	R&G
prediction	time	$O(m^6)$	$O(m^4)/O(m^5)$	$O(m^5)$	$O(m^5)$	$O(m^5)$	$O(m^4)$
	space	$O(m^4)$	$O(m^3)/O(m^3)$	$O(m^3)$	$O(m^4)$	$O(m^4)$	$O(m^2)$

We have seen now that for a structure prediction algorithm following our scheme both the algorithmic complexity and the class of structures that it is able to predict directly depend on the associated set of split types $\mathcal{T}$. Now we identify those split types for the various existing pseudoknot prediction algorithms.

3.2.3 Instances of the Scheme

In contrast to pseudoknot alignment, for pseudoknot structure prediction various algorithms have been developed [31, 46, 56, 1, 16, 17, 9, 44]. The complexities of the different algorithms are shown in Table 3.1. Each algorithm has a corresponding class of pseudoknots that it is able to predict. Following Condon *et al.* [11], we name the classes of structures according to the authors of the respective prediction algorithms: R&E (Rivas and Eddy [46]), A&U (Akutsu [1] and Uemura [56]), L&P (Lyngsø and Pedersen [31]), D&P (Dirks and Pierce [17]), CCJ (Chen, Condon, and Jabbari [9]), and R&G (Reeder and Giegerich [44]). For each of them, we now show how to interpret the respective algorithm as an instance of our algorithm scheme. Furthermore, we show for each of them how to explain the time and space complexity with the help of the results given in the previous section. For the class of simple recursive pseudoknots we even show how to improve the space complexity from $O(n^4)$ to $O(n^3)$ compared to the original algorithm of Akutsu [1].

R&E structures

The algorithm by Rivas and Eddy [46] recursively considers all possible fragments with a degree of at most two. The fragments are recursively computed according to the recursion scheme given in (3.1) with all split types for which the parent and both children have a degree of at most two. By Lemma 1 this results in a time and space complexity of $O(n^6)$ and $O(n^4)$ which is exactly the complexity of the R&E algorithm. This is the highest complexity among all algorithms we are going to consider and the class of generated structures is also the largest in this algorithm. All other algorithms can be considered as attempts to restrict the set of considered split types of the R&E algorithm to obtain a better complexity at the cost of a smaller set of covered structures.

A&U structures (simple and simple recursive pseudoknots)

The algorithms of Akutsu [1] and Uemura [56] are just two representations of the same algorithm. The algorithm of Deogun *et al.* [16] also just differs from those in its scoring scheme but not in its recursive nature. We consider the representation given by Akutsu. This algorithm is described in two variants: one for *simple pseudoknots* and one for *simple recursive pseudoknots*. Both compute matrices $S_L(i,j,k)$, $S_R(i,j,k)$ and $S_M(i,j,k)$ for all possible starting points i_0.[2] In our notation this corresponds to computing values for fragments of the form $([i_0,i],[j,k])$. The inspection of the recursions for S_L, S_R, and S_M shows that they employ the following split types:

$$\begin{array}{cccccc} 12'G2'1 & 1G2'12' & 12'G1 & 1G2'1 & 1G12' & 12'G2' \\ & & (12G1) & (1G21) & (1G12) & \end{array}$$

The split types in brackets are only used for simple recursive pseudoknots. The A&U algorithm for simple pseudoknots constructs the structures by recursively adding single, unpaired bases and single arcs. The second child in the split types represents a base, if it consists of a single, length constrained interval and an arc, if it consists of two length constrained intervals. The gapped fragments are combined to fragments without gaps (represented by the recursions for $S_{\text{pseudo}}(i_0,k_0)$ and $S(i,j)$ in the original presentation in [1]) with the split types

$$121 \qquad 12$$

By Lemma 2 this yields a time complexity of $O(n^4)$ for simple and $O(n^5)$ for simple recursive pseudoknots which is exactly the complexity reported in [1]. For example, we have $\#_C^n(12'G1) \in O(n^4)$ and $\#_C^n(12G1) \in O(n^5)$. Without any further optimization, also by Lemma 2 the space complexity is $O(n^4)$, but for simple pseudoknots Akutsu describes an evaluation order that reduces the space requirement to $O(n^3)$. This evaluation order does not work for simple recursive pseudoknots since it requires to compute all fragments of degree 2 before all fragments of degree 1 which is not possible with the additional split types required for simple recursive pseudoknots. We present now another evaluation order that also reduces the space complexity to $O(n^3)$ and works for both simple and simple recursive pseudoknots. This effectively leads to an improved algorithm for simple recursive pseudoknots with a better space complexity than known so far.

For a sequence of length n, we define the grouping on its fragments F as

$$\mathcal{S}_i := \left\{ F \,\middle|\, F \text{ has degree 1 and } F[1]^{\mathrm{L}} = n+1-i \right\} \tag{3.9}$$

$$\mathcal{G}_i := \left\{ F \,\middle|\, F \text{ has degree 2 and } F[1]^{\mathrm{L}} = n+1-i \right\} \tag{3.10}$$

By this grouping, all fragments are computed according to their leftmost boundary from right to left. Since all split types used by the algorithm, including the ones for simple recursive structures, are invariant with respect to this grouping, by Lemma 3 the computation requires $O(n^3)$ space.

[2] Actually the algorithm is presented as computing these values for all possible i_0 and k_0 and later it is shown that the values are independent of k_0.

L&P structures

Lyngsø and Pedersen [31] predict certain pseudoknots in $O(n^5)$ time and $O(n^3)$ space. With respect to our algorithm scheme their approach is in particular interesting since it uses a sophisticated grouping to reduce the space complexity. The algorithm constructs the final structure out of two gapped fragments according to the split type

$$\downarrow 12121 \downarrow .$$

These two fragments are computed recursively based on the following split types.

$\downarrow 12'G2'1G1\downarrow$	$\downarrow 1G21G1\downarrow$	$\downarrow 12G1G1\downarrow$	$\downarrow 2G1G12\downarrow$	(with 2 gaps)
2'1G12'	21G1	1G12	1G2	(with 1 gaps)
2'12'	12			(without gaps)

As in the A&U algorithm, split types where the second child has length constraints correspond to the addition of single arc. By Lemma 2 the set of split types yields a time and space complexity of $O(n^5)$ and $O(n^4)$, respectively. Note that due to the maximality constraint only $O(n^4)$ fragments with two gaps are considered. The space complexity is reduced to $O(n^3)$ with the following grouping of the fragments F of the sequence:

$$\mathcal{S}_i := \begin{cases} \{\, F \,|F \text{ has degree 1 and } F \neq \{[1,n]\} \,\} & \text{if } i = 1 \\ \emptyset & \text{if } i > 1 \end{cases} \tag{3.11}$$

$$\mathcal{G}_i := \left\{ F \;\middle|\; \begin{array}{r} F \text{ has degree 2 and } F[1]^{\mathrm{R}} = i-1 \text{ or } F[2]^{\mathrm{L}} = i \\ \text{or } F \text{ has degree 3 and } F[2]^{\mathrm{R}} = i-1 \end{array} \right\} \tag{3.12}$$

This grouping computes at the beginning (in $\mathcal{S}_1$) all simple fragments which are exactly the ones of degree 1 except the final one covering the entire sequence. This is possible since fragments of degree 1 are only computed by the splits $2'12'$ and 12 and hence only recursively rely on atomic fragments and other simple fragments with degree 1.

Then all grouped fragments are computed, where in this case the groups overlap. Any group $\mathcal{G}_i$ contains the fragments with two gaps where the last gap starts at $i+1$ and the fragments with one gap where this gap starts at $i+1$ or ends at i. In all the split types the start and end point of the last gap remains invariant among the parent and the first child. This can be seen in the pattern $1G1$ that is part of all the gapped split types used by the algorithm. Hence the Requirement (3.3) is satisfied for the first child. The second child is always atomic, except for the type $\downarrow 2G1G12\downarrow$. For $\downarrow 2G1G12\downarrow$ due to the maximality constraint the second child has only $O(n^2)$ possible instances and those are computed in a preprocessing step analogous to the recursion of fragments with degree one. Lyngsø and Pedersen motivate this with the idea that the sequence is cyclic. In the cyclic sequence, the second child in $\downarrow 2G1G12\downarrow$ has no gap but just consists of one contiguous area due to the maximality constraint. Each fragment of degree 1 or 3 is contained in

exactly one group and each fragment of degree 2 is contained in two groups. Hence Requirement (3.4) is also satisfied.

The grouping does not yet include the final fragment $\{[1, n]\}$. The reason is that this is calculated with the split type 12121 step by step in-between the different stages applied by the grouping and hence does not fit exactly in the scheme. Once the values for some group $\mathcal{G}_i$ are computed, the instances of 12121 for which the two children are contained in $\mathcal{G}_i$ are considered. Due to the construction of the grouping, both children are always contained in the same group: if the gap of the second child ends at i the last gap of the first child starts at $i+1$ since those are directly adjacent in 12121.

D&P structures

Dirks and Pierce [17] developed an algorithm to compute the partition function for RNA pseudoknots that can also be modified to predict the MFE structure. The main restriction compared to the general R&E algorithm is that the D&P algorithm does not consider splits where both children and the parent are unconstrained and have a degree of two. This is reasonable since those are the only ones that have $O(n^6)$ instances for the children. In detail the considered split types are

$$12 \quad 1212 \quad 21\mathrm{G}1 \quad 12\mathrm{G}1 \quad 1\mathrm{G}21 \quad 1\mathrm{G}12 \quad 1'2G21'.$$

The complexity of $O(n^5)$ time and $O(n^4)$ space follows directly from Lemma 2 without any further grouping or other optimizations.

CCJ structures

The algorithm of Chen *et al.* [9] is able to predict the class of CCJ structures, which is a class containing for example kissing hairpins and 4-chains (i.e. four stems arranged from left to right such that each stem overlaps with the neighboring stems). CCJ structures are recursively composed out of TGB fragments (three groups of bands). A TGB fragment is a structure with one gap as visualized in Figure 3.4. It can be parsed with the following split types:

$2'12'G1$	$2'1G12'$	$12'G2'1$	$1G2'12'$	(add a single arc)
21G1	12G1	1G21	1G12	(add a recursive CCJ structure)

A CCJ structure is composed from TGB fragments by the following split types

1212		(combine two TGB structures)
$1'21'$	12	(add arc or concatenate two structures)

For all those split types T for TGB and CCJ structures, by Lemma 2 $\#_C^m(T) \in O(m^5)$ and $\#_P^m(T) \in O(n^4)$. Hence the algorithm runs in $O(n^5)$ time and $O(n^4)$ space.

The recursions for TGB fragments can be considered as an extension of the A&U recursions. The three types $2'12'G1$, $2'1G12'$, and $21G1$ are added which at the one

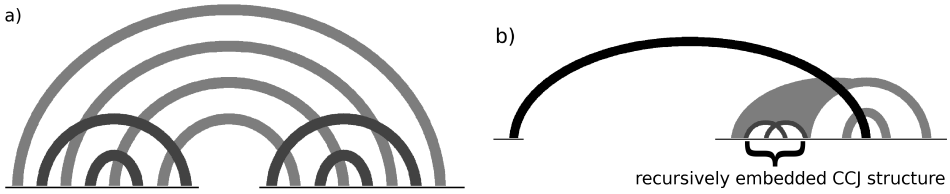


Figure 3.4: a) Schematic representation of a TGB fragment. It consists of three groups of bands of which the middle one spans the gap. b) TGB structures may contain CCJ structures recursively (Figures taken from Chen *et al.* [9]).

hand allows to generate more complex structures (with three instead of two groups of bands) but on the other hand destroys the invariant that is necessary to optimize the space complexity to $O(n^3)$.

R&G structures

The efficiency of the Reeder and Giegerich [44, 45] structure prediction algorithm ($O(n^4)$ time, $O(n^2)$ space) is due to the restriction to canonical pseudoknots. A stem of base pairs is called *canonical* if it cannot be extended by another valid base pair. The canonical stem containing a given base pair is thus uniquely determined. In R&G structures, pseudoknots are formed only by two crossing canonical stems. An example for such a canonical pseudoknot is given in Figure 3.5.

The Reeder and Giegerich recursion uses the following split types

$$2'1 \quad 12' \quad 12 \quad 1'21' \quad 1^c23^c41^c53^c$$

where the c's in the last type indicate a new kind of constraint, namely that the respective stem is canonical. Also note that this last type contains five instead of two children. In order to simplify the presentation, so far we have restricted splits to have only two children but the problem generalizes to k children without problems.

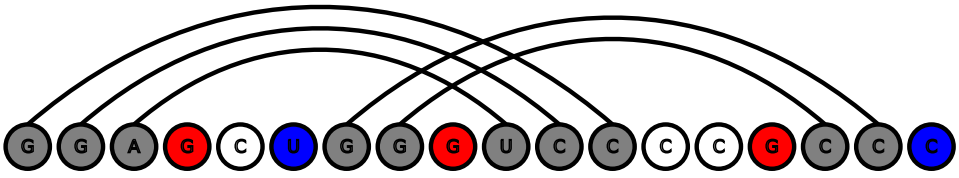


Figure 3.5: An example for a canonical pseudoknot. Both stems are maximal in the sense that they cannot be extended with additional arcs. An extension with an inner arc is not possible for any of the two stems, since the red G's cannot pair, the left stem cannot be extended to the outside, since it already starts at the beginning of the sequence and the second stem cannot be extended to the outside since the blue U and C cannot pair.

The types of the R&G algorithm directly correspond to the grammar rules given in [44] to describe the algorithm in terms of the ADP framework [23]:

$$S \to . \mid . \; S \mid S. \mid SS \mid (S) \mid [^k S \{^l S]^k S\}^l$$

In this grammar, $[^k$ $]^k$ and $\{^l$ $\}^l$ denote the two canonical stems of length k and l, respectively, i.e. k and l denote that the respective symbol in the grammar should be repeated k and l times, respectively. Note that $\#_C^n(1^c23^c41^c53^c) \in O(n^4)$ and $\#_P^n(1^c23^c41^c53^c) \in O(n^2)$: the parent has no gap and the two children are uniquely determined by choosing two canonical (i.e. maximally extended stems) of which there are only $O(n^2)$ many for each of them. To see that there are $O(n^2)$ canonical stems, just note that you can choose arbitrarily the start and end point of some arc and then extend it to a maximal stem.

For all split types T other than $1^c23^c41^c53^c$, $\#_C^m(T) \in O(m^3)$ and $\#_P^m(T) \in O(n^2)$ follows from Lemma 2. Hence, the overall time and space complexity of the R&G algorithm is $O(n^4)$ and $O(n^2)$, respectively.

3.3 A General Framework for DP based Alignment of RNA Structures

In the last section we have seen that the concepts of parse trees and gapped fragments allow to explain all known DP based pseudoknot prediction algorithms in a common framework. We now develop a similar framework for DP based pseudoknot alignment. All pseudoknot prediction algorithms presented later in this thesis will be explained in terms of this framework. As a consequence the similarities and differences of the various algorithms are clearly visible. Furthermore, the design decisions taken for each of the algorithms become very explicit which leads to a thorough understanding of how these decisions affect the complexity of the respective algorithm and lead to certain restrictions on the input structures. The fact that also already existing alignment algorithms can be explained in terms of the framework both shows the generality of the approach and allows for a precise comparison with previous work.

Throughout this section we assume the presence of two arc-annotated sequences that should be aligned to each other and denote them as (S_a, P_a) and (S_b, P_b), respectively.

3.3.1 From Single Sequences to Sequence Pairs

While a parse tree represents a recursive decomposition of a single RNA structure, for pairwise alignments we are always concerned with two structures. The central idea for DP based alignment methods is to compose the final alignment out of optimal alignments of certain fragments. Hence, both structures have to be decomposed simultaneously in the same way. This is formalized with the notion of fragment pairs and split pairs.

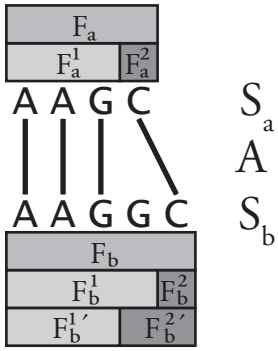


Figure 3.6: An alignment A that aligns the fragment pair (F_a, F_b). The figure shows one split (F_a^1, F_a^2) of F^a and two different possible splits (F_b^1, F_b^2) and $(F_b^{1'}, F_b^{2'})$ of F_b. Note that the split of F_a is aligned to both splits of F_b.

Definition 11 (fragment pair and split pair) A fragment pair of degree k is a pair $\mathcal{F} = (F_a, F_b)$, where F_a is a fragment of the first sequence (S_a, P_a) and F_b is a fragment of the second sequence (S_b, P_b) and both fragments have the same degree k. A split pair $(\mathcal{F}^1, \mathcal{F}^2)$ of $\mathcal{F}$ with basic split type T is a pair where $\mathcal{F}^1 = (F_a^1, F_b^1)$ and $\mathcal{F}^2 = (F_a^2, F_b^2)$ such that (F_a^1, F_a^2) is a split of F_a and (F_b^1, F_b^2) is a split of F_b and both splits have basic type T. □

Definition 12 (alignment of fragments and splits) An alignment A *aligns a fragment pair* $\mathcal{F} = (F_a, F_b)$ of degree k, short $\text{align}_A(\mathcal{F})$, if and only if for all $(a_1, a_2) \in A$ and for all $i \in 1 \ldots k$ it holds that $a_1 = -$ or $a_2 = -$ or $a_1 \in F_a[i] \Leftrightarrow a_2 \in F_b[i]$. Furthermore, A *aligns a split pair* $(\mathcal{F}^1, \mathcal{F}^2)$, if and only if $\text{align}_A(\mathcal{F}^1)$ and $\text{align}_A(\mathcal{F}^2)$. The *restriction of* A *to* $\mathcal{F}$ is defined as $A|_{\mathcal{F}} = A \cap ((\hat{F}_a \cup \{-\}) \times (\hat{F}_b \cup \{-\}))$. □

Note that for a given alignment A, a fragment (or a split) of one sequence can be aligned to several fragments (or splits) of the other. As an example consider Figure 3.6.

In general DP algorithms find a solution by recursively solving subproblems. In the case of alignment a subproblem can be restricted in the following two ways. Either a subproblem consists of finding the optimum only among a restricted set $\mathcal{A}$ of alignments and not among all possible alignments, or it consists of finding the optimal alignment only for certain fragments (F_a, F_b) of the sequences and not for the entire sequences. Both restrictions can be combined. The solution of such a restricted subproblem is represented by $C_{\mathcal{A}}(F_a, F_b)$ which is defined as follows.

Definition 13 ($C_{\mathcal{A}}(\mathcal{F})$) For any pair of fragments $\mathcal{F} = (F_a, F_b)$ and any set $\mathcal{A}$ of alignments such that for each $A \in \mathcal{A}$ is holds $\text{align}_A(\mathcal{F})$, we define

$$C_{\mathcal{A}}(\mathcal{F}) := \min_{A \in \mathcal{A}} \text{cost}(A|_{\mathcal{F}})$$

and furthermore

$$C(\mathcal{F}) := C_{\mathcal{A}}(\mathcal{F}) \text{ for } \mathcal{A} = \{ A \,|\text{align}_A(\mathcal{F}) \} \qquad \square$$

Intuitively, $C_{\mathcal{A}}(\mathcal{F})$ represents the optimal cost to align the fragment pair $\mathcal{F}$ under the assumption that the optimal alignment is contained in the set of alignments $\mathcal{A}$ and under the assumption that this optimal alignment contains no open arc matches for $\mathcal{F}$.

Definition 14 (open arc matches) Let A be an alignment that matches some fragment pair $\mathcal{F} = (F_a, F_b)$. A pair of arcs $(p_a, p_b) \in P_a \times P_b$ is called an *open arc match* for (F_a, F_b), if and only if (p_a, p_b) is matched by A and p_a is open in F_a, i.e. $p_a^{\mathrm{L}} \in \hat{F}_a \not\Leftrightarrow p_a^{\mathrm{R}} \in \hat{F}_a$. $\square$

Since A matches the two arcs as well as the two fragments, the fact that p_a is open in F_a is equivalent to the fact that p_b is open in F_b. If both F_a and F_b are arc complete the fragment pair (F_a, F_b) has no open arc matches. Note that the converse does not hold, since F_a and F_b might have open arcs that are not matched to any arcs in the other sequence.

3.3.2 The Basement for Recursive Computation of Alignments

In the previous section we have defined the general notion of $C_{\mathcal{A}}(\mathcal{F})$ that corresponds to the optimal alignment of a fragment pair $\mathcal{F}$ provided that the optimal alignment is contained in the set of alignments $\mathcal{A}$. A DP based alignment algorithm can now be described as recursively computing such entries $C_{\mathcal{A}}(\mathcal{F})$.

While this is common to all alignment algorithms we consider, the main difference is which instances of $C_{\mathcal{A}}(\mathcal{F})$ are considered and in what way they are recursively computed. For those aspects there is a wide range of choices but nevertheless, all of them have to obey certain rules to guarantee the correctness of the algorithm. In this section we present some basic lemmata that capture those general rules and show how $C_{\mathcal{A}}(\mathcal{F})$ can be computed recursively.

The lemmata build the basement for all alignment algorithms presented later and probably also for most other DP based RNA alignment algorithms. They capture simple properties that should not be regarded as surprising new facts but rather as the central properties of the problem domain. Making these properties explicit allows to base the correctness proofs of the algorithms presented later on a proper core. It also simplifies the process to adapt the algorithms to other related problems (as alignments for different kinds of structures or for different scoring schemes), since many properties directly follow once it is shown that the core lemmata are satisfied.

Lemma 4 (case distinction) *Let $\mathcal{F}$ be a fragment pair, $\mathcal{A}$ a set of alignments that align $\mathcal{F}$ and $\mathcal{A}_1, \ldots, \mathcal{A}_k$ subsets of $\mathcal{A}$ such that $\bigcup_{i \in [1,k]} \mathcal{A}_i = \mathcal{A}$. Then*

$$C_{\mathcal{A}}(\mathcal{F}) = \min_{i \in [1,k]} C_{\mathcal{A}_i} \qquad \square$$

PROOF

$$C_{\mathcal{A}}(\mathcal{F}) = \min_{A\in\mathcal{A}} \text{cost}(A|_{\mathcal{F}}) = \min_{i\in[1,k]} \min_{A\in\mathcal{A}_i} \text{cost}(A|_{\mathcal{F}}) = \min_{i\in[1,k]} C_{\mathcal{A}_i}$$

■

This simple lemma shows in the most general way what choices are available to traverse the search space of all possible alignments to find the optimal one. The lemma alone is sufficient for a naive generate and test algorithm (by choosing $\mathcal{A}_1, \ldots, \mathcal{A}_k$ to be all possible singleton sets). To obtain more efficient algorithms, the case distinction must be combined with a decomposition into alignments of independent fragments. Since in our scoring scheme the only cost that is not local to a single position is the match of two arcs, the notion of independence is captured in the absence of open arc matches. If the absence of open arc matches can be guaranteed, an alignment problem can be decomposed into independent subproblems according to the following lemma.

Lemma 5 (independence) *Let $\mathcal{A}$ be a set of alignments and (F_a, F_b) be a fragment pair such that for all $A \in \mathcal{A}$ the fragment pair (F_a, F_b) is aligned by A and contains no open arc matches. Furthermore, let $((F_a^1, F_a^2), (F_b^1, F_b^2))$ be a split pair of (F_a, F_b) such that both both (F_a^1, F_b^1) and (F_a^2, F_b^2) also satisfy those properties (i.e. are aligned by all A and contain no open arc matches). Then*

$$C_{\mathcal{A}}(F_a, F_b) \geq C_{\mathcal{A}}(F_a^1, F_b^1) + C_{\mathcal{A}}(F_a^2, F_b^2) \geq C(F_a, F_b)$$

Furthermore, if $\mathcal{A}$ contains the optimal alignment then

$$C_{\mathcal{A}}(F_a, F_b) = C_{\mathcal{A}}(F_a^1, F_b^1) + C_{\mathcal{A}}(F_a^2, F_b^2)$$

□

PROOF

$$\begin{aligned}
C_{\mathcal{A}}(F_a, F_b) &\overset{\text{Def.13}}{=} \min_{A\in\mathcal{A}} \text{cost}(A|_{F_a\times F_b}) && (3.13)\\
&= \min_{A\in\mathcal{A}} \left(\text{cost}(A|_{F_a^1\times F_b^1}) + \text{cost}(A|_{F_a^2\times F_b^2})\right) && (3.14)\\
&\geq \min_{A\in\mathcal{A}} \text{cost}(A|_{F_a^1\times F_b^1}) + \min_{A\in\mathcal{A}} \text{cost}(A|_{F_a^2\times F_b^2}) && (3.15)\\
&\overset{\text{Def.13}}{=} C_{\mathcal{A}}(F_a^1, F_b^1) + C_{\mathcal{A}}(F_a^2, F_b^2) && (3.16)
\end{aligned}$$

Line 3.14 holds due to the definition of cost (Definition 3) and the absence of open arc matches in (F_a^1, F_b^1) and (F_a^2, F_b^2). To show Line 3.15 consider alignments $A_1, A_2 \in \mathcal{A}$ for which $cost(A_1|_{F_a^1\times F_b^1}) = C_{\mathcal{A}}(F_a^1, F_b^1)$ and $cost(A_2|_{F_a^2\times F_b^2}) = C_{\mathcal{A}}(F_a^2, F_b^2)$. Then for $A' := A_1 - A_1|_{F_a^2\times F_b^2} \cup A_2|_{F_a^2\times F_b^2}$, it holds that $cost(A'|_{F_a,F_b}) = cost(A_1|_{F_a^1,F_b^1}) + cost(A_2|_{F_a^2,F_b^2})$ and hence $C_{\mathcal{A}}(F_a^1, F_b^1) + C_{\mathcal{A}}(F_a^2, F_b^2) = C_{\{A'\}}(F_a, F_b) \geq C(F_a, F_b)$. If $\mathcal{A}$ contains the optimal alignment, $C_{\mathcal{A}}(F_a, F_b) = C_{\mathcal{A}}(F_a^1, F_b^1) + C_{\mathcal{A}}(F_a^2, F_b^2)$ follows directly from $C_{\mathcal{A}}(F_a, F_b) = C(F_a, F_b)$. ■

In the case that $C_{\mathcal{A}}(F_a, F_b) > C_{\mathcal{A}}(F_a^1, F_b^1) + C_{\mathcal{A}}(F_a^2, F_b^2)$, an alignment A' that satisfies $C_{\{A'\}}(F_a, F_b) = C_{\mathcal{A}}(F_a^1, F_b^1) + C_{\mathcal{A}}(F_a^2, F_b^2)$ is not contained in $\mathcal{A}$ but can easily be constructed as described in the proof of Lemma 5. To simplify the presentation, we hence assume from now on that $C_{\mathcal{A}}(F_a, F_b) = C_{\mathcal{A}}(F_a^1, F_b^1) + C_{\mathcal{A}}(F_a^2, F_b^2)$. Formally this can be justified by implicitly replacing $C_{\mathcal{A}}(F_a, F_b)$ by $C_{\mathcal{A}\cup\{A'\}}(F_a, F_b)$ in cases where $A' \notin \mathcal{A}$. Since in all algorithms we are finally interested in the optimum among all possible alignments, adding A' is always safe. The next lemma captures the fact that $C_{\mathcal{A}}(\mathcal{F})$ depends only on the part of the alignments in $\mathcal{A}$ that is covered by $\mathcal{F}$.

Lemma 6 (cost locality) *Let $\mathcal{F}$ be a fragment pair and $\mathcal{A}_1$ and $\mathcal{A}_2$ be sets of alignments such that for each $A \in \mathcal{A}_1 \cup \mathcal{A}_2$ is holds that* $\text{align}_A(\mathcal{F})$ *and furthermore*

$$\{\ A|_{\mathcal{F}} |\ A \in \mathcal{A}_1\ \} = \{\ A|_{\mathcal{F}} |\ A \in \mathcal{A}_2\ \} \tag{3.17}$$

Then

$$C_{\mathcal{A}_1}(\mathcal{F}) = C_{\mathcal{A}_2}(\mathcal{F}) \qquad \square$$

Proof

$$C_{\mathcal{A}_1}(\mathcal{F}) \overset{\text{Def.13}}{=} \min_{A\in\mathcal{A}_1} \text{cost}(A|_{\mathcal{F}}) \overset{\text{Eq.3.17}}{=} \min_{A\in\mathcal{A}_2} \text{cost}(A|_{\mathcal{F}}) \overset{\text{Def.13}}{=} C_{\mathcal{A}_2}(\mathcal{F}) \tag{3.18}$$

■

As a corollary, it suffices to consider only alignments that contain no open arc matches for the considered fragment.

Definition 15 (arc complete cost) Let $C_{AC}(\mathcal{F}) := C_{\mathcal{A}}(\mathcal{F})$ for $\mathcal{A} = \{\ A\,|A$ aligns $\mathcal{F}$ and contains no open arc matches for $\mathcal{F}\ \}$. $\square$

Corollary 1 (arc complete costs) *For all $\mathcal{F}$ it holds that $C(\mathcal{F}) = C_{AC}(\mathcal{F})$.* $\square$

Proof The set of all alignments and the set of all arc complete alignments satisfy the properties required for $\mathcal{A}_1$ and $\mathcal{A}_2$ in Lemma 6. ■

4 Fixed Parameter Tractable Alignment of Arbitrary Pseudoknots

In this section several alignment algorithms for the alignment of two arc annotated sequences are considered. We start with a well known algorithm for plain sequence alignment and extend it step by step to an algorithm that can handle arbitrary pseudoknots. Each of the intermediate steps forms an algorithm that either improves the time and space complexity compared to the previous one or is able to align more complex structures. In that way we go from an algorithm for plain sequences without structure to an algorithm for pseudoknot free structures and finally several algorithms for the alignment of arbitrary pseudoknots.

Since each algorithm is presented as an extension of the previous ones, the similarities of the algorithms become apparent. This is also supported by the fact that all the algorithms are presented in terms of the general alignment framework developed in Section 3.3. As a further benefit, the framework also allows to give correctness proofs that are both precise and simple.

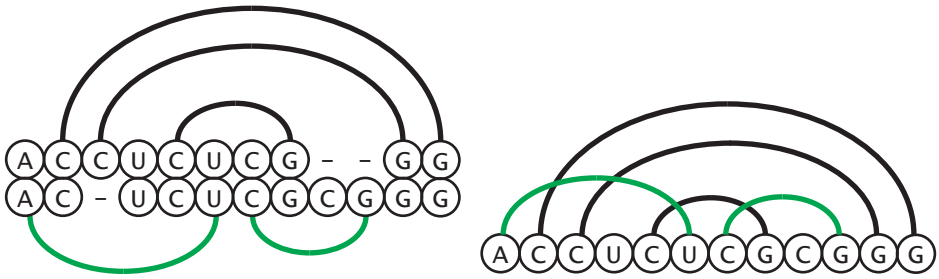
Throughout the chapter we will denote the two arc annotated sequences that are aligned by the algorithms with (S_a, P_a) and (S_b, P_b). Furthermore, n and m denote the length of the two sequences, respectively. Most algorithms presented in this chapter only consider fragments of degree 1. To simplify notation, we will therefore represent a fragment $([i, i'])$ as $[i, i']$. Before we look at the algorithms we take a look at the theoretical complexity of the problem that the algorithms solve.

4.1 Hardness Results for the Alignment of Pseudoknots

In this section first some complexity results for sequence structure alignment are reviewed. In particular the NP-completeness of the alignment of pseudoknots is shown for various scoring schemes. Then the concept of parameterized complexity is explained. This concept allows a more fine grained analysis of the complexity of NP-hard problems and will be used to show that the algorithms presented later in this chapter are fixed parameter tractable.

4.1.1 NP-Hardness Results

The complexity of an alignment problem mainly depends on two aspects: the class of considered structures and the scoring scheme. These two aspects are captured in the alignment hierarchy of Blin *et al.* [7]. This hierarchy considers three different models for the scoring scheme that are all based on some edit distance:



(a) alignment (b) common superstructure

Figure 4.1: An alignment of two nested structures with a crossing superstructure.

- model I allows arbitrary substitutions, base-deletions and arc-removings,
- model II allows arc-altering and all operations of model I, and
- model III allows arc-breakings and all operations of model II.

Concerning the structures, the alignment hierarchy distinguished between nested structures (NEST), crossing structures (CROSS) and unlimited structures (UNLIM). The latter also allow many arcs to be adjacent to the same base. Besides the structure of the two input sequences, the alignment hierarchy also considers the common superstructure. This structure is obtained by overlaying the two input structures according to the alignment; an example is given in Figure 4.1. A restriction on the class of superstructures can be thought of as a constraint on the optimal alignment that is computed. If the superstructure is for example required to be nested, the algorithm computes the optimal alignment only among all alignments that do not lead to a crossing superstructure.

An instance of the alignment hierarchy is described by a scoring scheme model $M \in \mathrm{I}, \mathrm{II}, \mathrm{III}$ and three structure classes A, B, $C \in \{\text{NEST}, \text{CROSS}, \text{UNLIM}\}$. A and B denote the structure classes for the two input structures and C denotes the structure class for the common superstructure, as we indicate with the notation $A \times B \to C$.

The known complexity results for the instances of the alignment hierarchy are summarized in Table 4.1. One of the most striking aspects of those results is that polynomial complexities have only been identified for problems where at least one of the input structures is nested. As soon as both structures are crossing the problem becomes NP-complete or even Max SNP-hard. The latter means that unless P=NP their does neither exist a polynomial time algorithm nor a polynomial time approximation scheme for the problem.

Concerning the different scoring models Table 4.1 shows that the arc-altering and arc-breaking operations of models II and III make the problem harder. While model I is solvable in polynomial time as long as at least one structure is nested, models II and III can only be solved in polynomial time if both input structures and the common super structure are nested.

Table 4.1: Complexity results for the alignment hierarchy according to Blin *et al.* [7].

structures	scoring schemes		
	I	II	III
NEST×NEST→NEST	$O(n^4)$	$O(n^4)$	$O(n^4)$
NEST×NEST→CROSS	$O(n^3 \log(n))$	NP-complete	
NEST×NEST→UNLIM	$O(n^3 \log(n))$	NP-complete	NP-complete
CROSS×NEST→CROSS	$O(n^3 \log(n))$	NP-complete	
CROSS×NEST→UNLIM	$O(n^3 \log(n))$	NP-complete	Max SNP-hard
CROSS×CROSS→CROSS	NP-complete	NP-complete	
CROSS×CROSS→UNLIM	NP-complete	NP-complete	Max SNP-hard
UNLIM×NEST→UNLIM	$O(n^3 \log(n))$	NP-complete	Max SNP-hard
UNLIM×CROSS→UNLIM	NP-complete	NP-complete	Max SNP-hard
UNLIM×UNLIM→UNLIM	NP-complete	NP-complete	Max SNP-hard

An interesting special case for a scoring scheme is the *general edit distance* of Jiang *et al.* [27] (described in Section 2.2.2). It corresponds to model III but with the additional restriction that an arc-altering must always have the cost of half an arc-breaking and half an arc-deletion. This additional restriction turns out to be crucial for the complexity of the problem and allows, for example, to solve CROSS×NEST→CROSS in $O(n^4)$ time and $O(n^2)$ space. Since the general edit distance subsumes the scoring model I, in the case of two crossing input structures the problem becomes NP-hard.

Among the considered scoring schemes the general edit distance is the most expressive one that is solvable in polynomial time at least for non-crossing input structures. Therefore we focus on it in this thesis. Since for arbitrary pseudoknots a polynomial time bound is not achievable we will analyze the algorithms by means of parameterized complexity.

4.1.2 There is still Hope: Parameterized Complexity

As described in the previous section, in the presence of arbitrary pseudoknots the alignment problem is NP-hard. Hence, unless P=NP there is no hope to find a polynomial runtime guarantee. Therefore we analyze the time and space requirements of the algorithms for arbitrary pseudoknots not in terms of classical complexity theory but in terms of parameterized complexity [18].

In contrast to classical complexity theory, parameterized complexity measures the complexity as a function that does not only depend on the size of the input but also on some additional parameter. One can think of this parameter as an additional aspect or property of the input besides its size. More formally, the introduction of such a parameter partitions the set of problem instances into k-slices such that each k-slice contains the instances for which the parameter equals k. Whereas in classical complexity theory the complexity is measured by a function $f(n)$ that is an upper bound for the runtime of all instances of size n, in parameterized complexity

a function $f(n, k)$ gives an upper bound for the runtime of only the instances of size n that are contained in the respective k-slice.

If this parameter is appropriately chosen, many problems turn out to be only exponential in the parameter k but polynomial in n. In that sense a parameterized complexity analysis is able to identify that the worst case exponential runtime of some algorithm is not directly related to the size of the input but only to some other aspect that might grow arbitrarily with the input but not necessarily does in practice.

The main goal when giving a runtime guarantee in terms of parameterized complexity is to identify a parameter that it small on practical instances. If for the instances of interest the parameter is bounded by some constant, for those instances the algorithm has a polynomial runtime (according to the classical measure in terms of input size). An algorithm for which this is the case is called fixed parameter tractable.

Concerning the alignment of pseudoknots there exists already an encouraging example for a fixed parameter tractable algorithm. This algorithm by Evans [20] computes the *longest arc-preserving common subsequence* (LAPCS) of two pseudoknot structures. The LAPCS problem corresponds to an instance of the scoring model II of the alignment hierarchy [7] and hence is NP-complete (see Table 4.1). However, in terms of parameterized complexity the algorithm by Evans runs in $O(9^k nm)$ time, where n and m represent the length of the two input structures, respectively, and k is the maximum *arc cutwith* among all sequence positions. The arc cutwith of a sequence position i is defined as the number of arcs (j, k) for which $j < i \leq k$. This number is usually much smaller that the sequence length.

4.2 Relevant Subproblems that are Solvable in Polynomial Time

Before looking at the alignment of arbitrary pseudoknot structures we take a look at alignment problems for more restricted structure classes that are solvable in polynomial time. More precisely, we look at two well known algorithms for the alignment of plain sequences and nested structures and show how they fit into the general alignment framework presented in the previous chapter. The algorithm for nested structures is presented as an extension of the algorithm for plain sequences. This is in particular interesting since later in this chapter the algorithm for nested structures is extended similarly to handle pseudoknots.

In each extension, the main goal is to use the recursions of the simpler algorithms as much as possible and only to switch to a more complex, slower recursion for parts in the structure, where it is really necessary. Therefore, a thorough understanding of the alignment methods presented in this section is necessary to understand the pseudoknot alignment methods following later.

4.2.1 Alignment of Plain Sequences

In the absence of secondary structure in at least one of the two sequences (i.e. if $P_1 = \emptyset$ or $P_2 = \emptyset$) the alignment can be computed according to the well known Needleman Wunsch algorithm [39]. This algorithm recursively computes the alignment of all possible prefixes of the two sequences. Since the same recursion will later be required also to compute other fragments of the sequences, the following lemma describes the recursion in a more general form for fragments $[i, i']$, although it suffices to consider fragments of the form $[1, i']$ for plain sequence alignment.

Lemma 7 (plain sequence alignment) *Let $P_1 = \emptyset$ or $P_2 = \emptyset$ and i, i', j, i' such that $1 \le i < i' \le |S_a|$ and $1 \le j < j' \le |S_b|$. Then*

$$C([i,i'],[j,j']) = \min \begin{cases} C([i,i'-1],[j,j']) + \mathrm{gap}_1(i') & \text{(4.1a)} \\ C([i,i'],[j,j'-1]) + \mathrm{gap}_2(j') & \text{(4.1b)} \\ C([i,i'-1],[j,j'-1]) + \mathrm{basematch}(i',j') & \text{(4.1c)} \end{cases}$$

□

Based on the lemmata developed in Section 3.3 we can easily prove this recursion now. Although the recursion is well known and not very complex, the proof is given in detail, since it forms the basis of the proofs following later in the chapter.

Proof Consider the following partition of the set of alignments that align $[i, i']$ to $[j, j']$, i.e. of $\mathcal{A} = \{ A \,|\mathrm{align}_A([i,i'],[j,j']) \}$ into

$$\mathcal{A}_1 := \left\{ A \in \mathcal{A} \,\middle|\, (i',-) \in A \right\} \qquad \mathcal{A}_2 := \left\{ A \in \mathcal{A} \,\middle|\, (-,j') \in A \right\}$$
$$\mathcal{A}_3 := \left\{ A \in \mathcal{A} \,\middle|\, (i',j') \in A \right\}$$

Then

$$C([i,i'],[j,j']) \overset{\text{Lem.4}}{=} \min \begin{cases} C_{\mathcal{A}_1}([i,i'],[j,j']) \\ C_{\mathcal{A}_2}([i,i'],[j,j']) \\ C_{\mathcal{A}_3}([i,i'],[j,j']) \end{cases} \tag{4.2}$$

$$\overset{\text{Lem.5}}{=} \min \begin{cases} C_{\mathcal{A}_1}([i,i'-1],[j,j']) + C_{\mathcal{A}_1}([i'],[j'+1,j']) \\ C_{\mathcal{A}_2}([i,i'],[j,j'-1]) + C_{\mathcal{A}_2}([i'+1,i'],[j']) \\ C_{\mathcal{A}_3}([i,i'-1],[j,j'-1]) + C_{\mathcal{A}_2}([i'],[j']) \end{cases} \tag{4.3}$$

$$\overset{\text{Lem.6}}{=} \min \begin{cases} C([i,i'-1],[j,j']) + C_{\mathcal{A}_1}([i'],[j'+1,j']) \\ C([i,i'],[j,j'-1]) + C_{\mathcal{A}_2}([i'+1,i'],[j']) \\ C([i,i'-1],[j,j'-1]) + C_{\mathcal{A}_2}([i'],[j']) \end{cases} \tag{4.4}$$

$$\overset{\text{Def.13}}{=} \min \begin{cases} C([i,i'-1],[j,j']) + \mathrm{gap}_1(i') \\ C([i,i'],[j,j'-1]) + \mathrm{gap}_2(j') \\ C([i,i'-1],[j,j'-1]) + \mathrm{basematch}(i',j') \end{cases} \tag{4.5}$$

Since $P_1 = \emptyset$ or $P_2 = \emptyset$ the absence of open arc matches required for Lemma 5 in Line 4.3 is trivially satisfied. ■

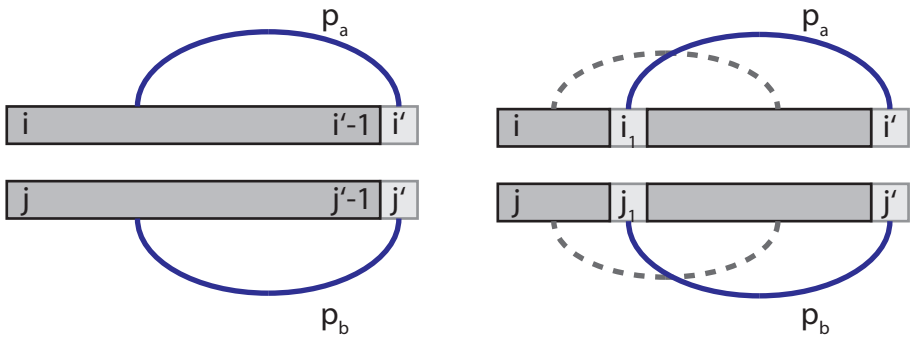


Figure 4.2: The recursion for plain sequence alignment is not correct in the presence of secondary structure, since it may cause open arc matches in the fragments that are recursively considered as shown in (a). Part (b) shows a solution for that for nested structures. Here the alignment can be decomposed according to case 4.6(d) into the arc match itself, the part before the arc match $([i, i_1 - 1], [j, j_1 - 1])$ and the part below it $([i_1 + 1, i' - 1], [j_1 + 1, j' - 1])$. These two parts are only independent for nested structures. In the presence of pseudoknots they may again contain open arc matches as indicated by the gray, dashed arcs.

A dynamic programming algorithm based on the recursion of Lemma 7 computes a matrix with values $C([1, i'], [1, j'])$ for all $0 \leq i \leq n$ and $0 \leq i \leq m$, where $n = |S_a|$, $m = |S_b|$. Note that the two left boundaries can be fixed to 1, since the boundaries i and j stay invariant during the recursion. Therefore, the computation of the optimal alignment, $C([1, |S_a|], [1, |S_b|])$, relies only on the optimal alignments of all prefixes of the two sequences.

The recursion is applied to all entries with $i' > 0$ or $j' > 0$. If either $i' = 0$ or $j' = 0$ one of the two prefixes is the empty prefix and hence its last position cannot be aligned to a gap or another base. In the recursion, those respective cases would rely on fragments $[1, -1]$ that are not well-defined. Therefore those cases of the recursion are then implicitly skipped. Note that this is correct, since in those cases the respective sets $\mathcal{A}_1$ or $\mathcal{A}_2$ and $\mathcal{A}_3$ in the proof of Lemma 7 are empty. Finally, the only base case that is not computed according to the recursion is the cost to align to empty prefixes which is set to $C([1, 0], [1, 0]) = 0$. As each entry can be computed in constant time, the DP algorithm based on this lemma requires $O(nm)$ time and space.

4.2.2 Nested Sequence Structure Alignment

In the presence of secondary structure, the absence of open arc matches required for Lemma 5 in Equation 4.3 is not always satisfied. A case where this is not satisfied is shown in Figure 4.2(a).

The algorithm by Jiang *et al.* [27] solves this problem for nested structures. This algorithm handles arc matches by decomposing the alignment into the part before the arc match and the part below it as shown in Figure 4.2(b). The full recursion is described in the following lemma.

Lemma 8 (nested structure alignment) *Let P_1 or P_2 be nested and i, i', j, j' such that $1 \leq i < i' \leq |S_a|$ and $1 \leq j < j' \leq |S_b|$. Then*

$$C([i,i'],[j,j']) = \min \tag{4.6}$$
$$\begin{cases} C([i,i'-1],[j,j']) + \text{gap}_1(i') & \text{(4.6a)} \\ C([i,i'],[j,j'-1]) + \text{gap}_2(j') & \text{(4.6b)} \\ C([i,i'-1],[j,j'-1]) + \text{basematch}(i',j') & \text{(4.6c)} \\ \textit{if } \exists (p_a,p_b) = ((i_1,i'),(j_1,j')) \in P_a \times P_b \textit{ for some } i_1, j_1 & \text{(4.6d)} \\ \quad C([i,i_1-1],[j,j_1-1]) + C([i_1+1,i'-1],[j_1+1,j'-1]) & \\ \quad + \text{arcmatch}(p_a,p_b) & \end{cases}$$

□

Proof If there does not exist an appropriate i_1 or j_1, the proof is analogous to the sequence alignment case. Otherwise consider the partition of

$$\mathcal{A} = \{\, A \,|\, A \text{ aligns } [i,i'] \text{ to } [j,j'] \text{ and contains no open arc matches for those fragments} \,\}$$

into

$$\begin{aligned} \mathcal{A}_1 &:= \{\, A \in \mathcal{A} \,|\, (i',-) \in A \,\} \\ \mathcal{A}_2 &:= \{\, A \in \mathcal{A} \,|\, (-,j') \in A \,\} \\ \mathcal{A}_3 &:= \{\, A \in \mathcal{A} \,|\, (i',j') \in A \text{ and } (i_1,j_1) \notin A \,\} \\ \mathcal{A}_4 &:= \{\, A \in \mathcal{A} \,|\, (i',j') \in A \text{ and } (i_1,j_1) \in A \,\} \end{aligned}$$

Note that due to Corollary 1 it suffices to consider no open arc matches for the fragment pair $([i,i'],[j,j'])$. For the partition we have

$$C([i,i'],[j,j']) \overset{\text{Cor.1}}{=} C_{AC}([i,i'],[j,j']) \tag{4.7}$$
$$\overset{\text{Lem.4}}{=} \min \begin{cases} C_{\mathcal{A}_1}([i,i'],[j,j']) \\ C_{\mathcal{A}_2}([i,i'],[j,j']) \\ C_{\mathcal{A}_3}([i,i'],[j,j']) \\ C_{\mathcal{A}_4}([i,i'],[j,j']) \end{cases} \tag{4.8}$$

$$\overset{\text{Lem.5}}{=} \min \begin{cases} C_{\mathcal{A}_1}([i,i'-1],[j,j']) + C_{\mathcal{A}_1}([i'],[j'+1,j']) \\ C_{\mathcal{A}_2}([i,i'],[j,j'-1]) + C_{\mathcal{A}_2}([i'+1,i'],[j']) \\ C_{\mathcal{A}_3}([i,i'-1],[j,j'-1]) + C_{\mathcal{A}_3}([i'],[j']) \\ C_{\mathcal{A}_4}(\{[i,i_1-1],[i_1+1,i'-1]\},\{[j,j_1-1],[j_1+1,j'-1]\}) \\ \quad + C_{\mathcal{A}_4}([i_1],[i'],[j_1],[j']) \end{cases} \tag{4.9}$$

$$\overset{\text{Lem.5}}{=} \min \begin{cases} C_{\mathcal{A}_1}([i,i'-1],[j,j']) + C_{\mathcal{A}_1}([i'],[j'+1,j']) \\ C_{\mathcal{A}_2}([i,i'],[j,j'-1]) + C_{\mathcal{A}_2}([i'+1,i'],[j']) \\ C_{\mathcal{A}_3}([i,i'-1],[j,j'-1]) + C_{\mathcal{A}_3}([i'],[j']) \\ C_{\mathcal{A}_4}([i,i_1-1],[j,j_1-1]) + C_{\mathcal{A}_4}([i_1+1,i'-1],[j_1+1,j'-1]) \\ \quad + C_{\mathcal{A}_4}([i_1],[i'],[j_1],[j']) \end{cases} \tag{4.10}$$

$$\overset{\substack{\text{Lem.6}\\ \text{Def.13}}}{=} \min \begin{cases} C([i,i'-1],[j,j']) + \text{gap}_1(i') \\ C([i,i'],[j,j'-1]) + \text{gap}_2(j') \\ C([i,i'-1],[j,j'-1]) + \text{basematch}(i',j') \\ C([i,i_1-1],[j,j_1-1]) + C([i_1+1,i'-1],[j_1+1,j'-1]) \\ \quad + \text{arcmatch}(p_a,p_b) \end{cases} \tag{4.11}$$

Note that Equation 4.10 relies on the absence of pseudoknots: Lemma 5 requires both $([i,i_1-1],[j,j_1-1])$ and $([i_1+1,i'-1],[j_1+1,j'-1])$ to contain no open arc matches, which is ensured since any matched arcs connecting the two fragments would cross the arcs (i_1,i') and (j_1,j'), respectively (as the gray dashed arcs in Figure 4.2(b)). Any other open arc matches would also be open arc matches of the parent fragments $([i,i'],[j,j'])$ which we excluded from the partition due to Corollary 1. ■

The base cases for the recursion are $C([i_1,i_1-1][j_1,j_1-1]) := 0$ for all i_1, j_1. Compared to the Needleman Wunsch algorithm the Jiang algorithm does not only compute alignment costs for prefixes of the two sequences. As visualized in Figure 4.3 the algorithm computes in addition a quadratic number of square matrices, one for the region below each possible arc match. More precisely, for each $p_a \in P_a$ and $p_b \in P_b$ the algorithm computes $C(p_a^{\text{L}}+1,i'],[p_b^{\text{L}}+1,j'])$ for $p_a^{\text{L}} < i' < p_a^{\text{R}}$ and $p_b^{\text{L}} < i' < p_b^{\text{R}}$. The matrices are computed one after the other in an order such that for p_a, p_b, p'_a, p'_b with $p_a^{\text{L}} < p_a'^{\text{L}} < p_a'^{\text{R}} < p_a^{\text{R}}$ and $p_b^{\text{L}} < p_b'^{\text{L}} < p_b'^{\text{R}} < p_b^{\text{R}}$ the matrix of p_a, p_b is computed after the one for p'_a, p'_b. From each matrix only the last value, $C(p_a^{\text{L}}+1,p_a^{\text{R}}-1],[p_b^{\text{L}}+1,p_b^{\text{R}}-1])$ is required for the computation of the other matrices, namely in recursion case (4.6d). Hence all other values can be discarded once the entire matrix is computed. Therefore the computation of the in total $O(n^2m^2)$ matrix entries requires $O(n^2m^2)$ time but only $O(nm)$ space.

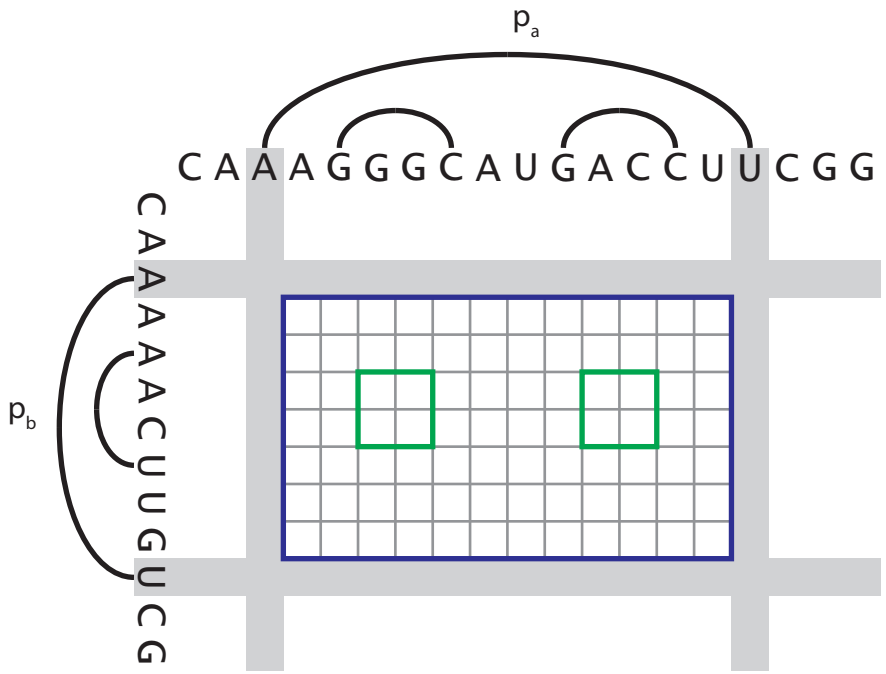


Figure 4.3: The Jiang algorithm computes one square matrix for each pair of arcs. The matrix marked by the blue rectangle contains the cost to align all prefixes of the two fragments below the arcs p_a and p_b. Due to recursion case 4.6(d) the computation of the blue matrix recursively relies on the last entries of the green matrices. Hence those are computed before the blue one.

4.3 From Nested Sequences to Arbitrary Pseudoknots

This section describes a new pseudoknot alignment algorithm published in [37]. The presentation in the paper differs significantly from this presentation here which describes the algorithm in terms of the general framework developed so far. This allows to highlight the similarities to the other algorithms described in this thesis and simplifies precise and clear correctness proofs.

The algorithm is described in three stages. First a basic variant is explained in Section 4.3.1 then, in Section 4.3.2 and 4.3.3, two extensions are presented that improve the time and space complexity.

4.3.1 A Basic Pseudoknot Algorithm

The algorithms presented so far are all based on the idea to recursively compose the optimal alignment out of alignments of fragment pairs without open arc matches. The recursion for plain sequence alignment (Lemma 7) is not suitable for nested structures since those may cause open arc matches as shown in Figure 4.2(a). Similarly the recursion for nested structures (Lemma 8) is not suitable for pseudoknots since those cause open arc matches as indicated with the gray dashed arc is Figure 4.2(b).

We now develop a general technique to handle any kind of open arc matches. In this section we apply the idea to extend the plain sequence alignment algorithm to cope with arbitrary pseudoknots. In the next section we then extend the algorithm for nested structures in an analogous way to obtain a more efficient algorithm.

The central idea is to handle each potential open arc match in a separate case that decomposes into this arc match and the remaining fragments without the arc match. Those remaining fragments then by construction have no open arc match anymore but they have gaps at the positions where the arc match is located. To describe fragment pairs containing such gaps the following notation is introduced.

Definition 16 (subtracting arc pairs from fragment pairs) Let (F_a, F_b) be a fragment pair and $M \subseteq P_1 \times P_2$ a set of (matched) arc pairs such that

I $\forall (p_a, p_b) \in M.\ p_a^{\mathrm{L}} \in \hat{F}_a \Leftrightarrow p_b^{\mathrm{L}} \in \hat{F}_b \Leftrightarrow p_a^{\mathrm{R}} \notin \hat{F}_a \Leftrightarrow p_b^{\mathrm{R}} \notin \hat{F}_b$

II $\exists A.\ \forall (p_a, p_b) \in M.\ (p_a^{\mathrm{L}}, p_b^{\mathrm{L}}) \in A \wedge (p_a^{\mathrm{R}}, p_b^{\mathrm{R}}) \in A$
(for some A, (p_a, p_b) is an arc match)

Then we denote with $(F_a, F_b) - M$ the fragment pair (F'_a, F'_b) where F'_a and F'_b are the fragments of minimal degree satisfying

$$\hat{F}'_a = \left\{ x \in \hat{F}_a \,\middle|\, \forall (p_a, p_b) \in M.\ p_a^{\mathrm{L}} \neq x \neq p_a^{\mathrm{R}} \right\} \tag{4.12}$$

$$\hat{F}'_b = \left\{ y \in \hat{F}_b \,\middle|\, \forall (p_a, p_b) \in M.\ p_b^{\mathrm{L}} \neq y \neq p_b^{\mathrm{R}} \right\} \tag{4.13}$$

$$\forall (p_a, p_b) \in M.\ \forall (x, y) \in \{(p_a^{\mathrm{L}}, p_b^{\mathrm{L}}), (p_a^{\mathrm{R}}, p_b^{\mathrm{R}})\}.\ \forall i. \tag{4.14}$$
$$F'_a[i]^{\mathrm{R}} < x < F'_a[i+1]^{\mathrm{L}} \Leftrightarrow F'_b[i]^{\mathrm{R}} < y < F'_b[i+1]^{\mathrm{L}}$$

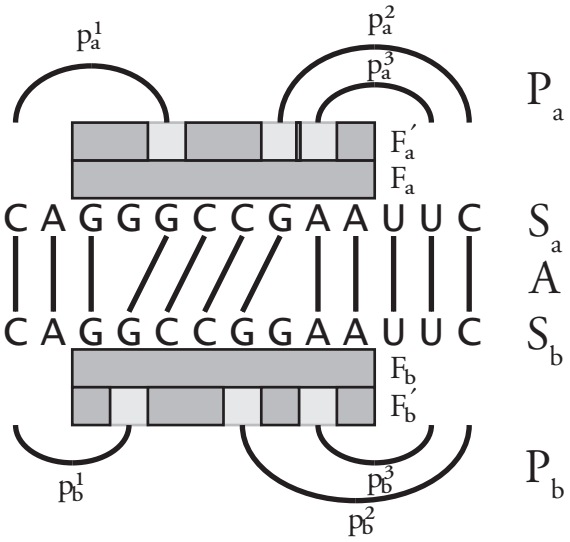


Figure 4.4: The fragments pair $(F_a, F_b) - M$ for $M = \{(p_a^1, p_b^1), (p_a^2, p_b^2), (p_a^3, p_b^3)\}$ is visualized as (F'_a, F'_b). Note that F'_a contains an interval of length 0 between the two left ends of p_a^2 and p_a^3.

If $(F_a, F_b) - M$ does not satisfy restriction I or II we call it invalid otherwise valid.□

An example for a fragment pair $(F_a, F_b) - M$ is given in Figure 4.4. Intuitively, $(F_a, F_b) - M$ denotes the fragment pair obtained by removing all positions of arc pairs in M from the fragment pair (F_a, F_b). Hence, if M is the set of open arc matches of (F_a, F_b) for some alignment A, then $(F_a, F_b) - M$ is a fragment pair that contains no open arc matches for A. A fragment pair $(F_a, F_b) - M$ is valid, if M is a potential set of open arc matches for (F_a, F_b). More precisely, each arc pair of M must have exactly one end contained in (F_a, F_b) (condition I) in order to be open for the fragment pair and furthermore it must be possible to match all arc pairs of M simultaneously (condition II).

The requirement 4.14 is a technical detail to ensure that the resulting fragments in both sequences have the same number of intervals. It states that if some end of an arc in the first sequence is located between the i-th and i plus first interval, the corresponding end of the arc in the second sequence must also be located there. This may cause empty intervals, if two arcs are directly adjacent in one of the two sequences. An example for this is given in Figure 4.4.

Now, we extend the recursion of the plain sequence alignment algorithm with special cases for open arc matches to obtain a recursion that is correct for arbitrary pseudoknots. Wherever the plain sequence alignment algorithm computes some cost $C([i, i'], [j, j'])$, the extended recursion computes values $C(([i, i'], [j, j']) - M)$ for all possible sets of open arc matches M.

Lemma 9 (basic pseudoknot alignment) *Let i, i', j, j' be such that $1 \leq i < i' \leq$*

$|S_a|$ and $1 \leq j < j' \leq |S_b|$ and let M be such that $([i,i'],[j,j']) - M$ is valid. Then

$$C(([i,i'],[j,j']) - M) = \min \tag{4.15}$$
$$\begin{cases} C(([i,i'-1],[j,j']) - M) + \text{gap}_1(i') & \text{(4.15a)} \\ C(([i,i'],[j,j'-1]) - M) + \text{gap}_2(j') & \text{(4.15b)} \\ C(([i,i'-1],[j,j'-1]) - M) + \text{basematch}(i',j') & \text{(4.15c)} \\ \textit{if } \exists (p_a,p_b) = ((i_1,i'),(j_1,j')) \in P_a \times P_b \textit{ for some } i_1 \geq i,\ j_1 \geq j & \text{(4.15d)} \\ \quad C(([i,i'-1],[j,j'-1]) - (M \cup \{(p_a,p_b)\}) + \text{arcmatch}(p_a,p_b) & \\ \textit{if } \exists (p_a,p_b) \in M \textit{ with } p_a^L = i',\ p_b^L = j' \textit{ or } p_a^R = i',\ p_b^R = j' & \text{(4.15e)} \\ \quad C(([i,i'-1],[j,j'-1]) - (M - \{(p_a,p_b)\})) & \end{cases}$$

In this recursion, cases referring to invalid items are implicitly skipped. □

The general idea of this algorithm is that the case where the optimal alignment has some set M of open arc matches for the fragment pair $([i,i'],[j,j'])$, is captured by an entry $C(([i,i'],[j,j']) - M)$ that by construction then has no open arc matches. The new last two cases of the recursion cover cases in which the set of open arc matches differs between the fragment pair $([i,i'],[j,j'])$ and its prefix $([i,i'-1],[j,j'-1])$. In case (4.15d) the arc match (p_a,p_b) is open in the prefix but not in $([i,i'][j,j'])$ and in case (4.15e) it is the other way around. The two situations are visualized in Figure 4.5. Another intuition is the following. The algorithm constructs the alignment from left to right and whenever a left end of an arc match is encountered, it is first introduced as a gap in the fragment pair via case (4.15e) . Later, when the right end is encountered, the gap is closed again by case (4.15d) which then adds the cost for the arc match.

Proof The proof follows the same scheme as the previous ones. This time consider the following partition of the set of alignments $\mathcal{A}$ that align $([i,i'],[j,j']) - M$ and contain no open arc matches for this fragment pair:

$$\mathcal{A}_1 := \{\, A \in \mathcal{A} \,|\, (i',-) \in A \,\}$$
$$\mathcal{A}_2 := \{\, A \in \mathcal{A} \,|\, (-,j') \in A \,\}$$
$$\mathcal{A}_3 := \left\{ A \in \mathcal{A} \;\middle|\; \begin{array}{l} (i',j') \in A \wedge \nexists (i_1,j_1) \in A. \\ \quad ((i_1,i'),(j_1,j')) \in P_a \times P_b \vee ((i',i_1),(j',j_1)) \in P_a \times P_b \end{array} \right\}$$
$$\mathcal{A}_4 := \left\{ A \in \mathcal{A} \;\middle|\; \begin{array}{l} (i',j') \in A \wedge \exists (i_1,j_1) \in A. \\ \quad ((i_1,i'),(j_1,j')) \in P_a \times P_b \wedge i_1 \geq i \wedge j_1 \geq j \end{array} \right\}$$
$$\mathcal{A}_5 := \left\{ A \in \mathcal{A} \;\middle|\; \begin{array}{l} (i',j') \in A \wedge \exists (i_1,j_1) \in A. \\ \quad ((i_1,i'),(j_1,j')) \in P_a \times P_b \wedge i_1 < i \wedge j_1 < j \quad \vee \\ \quad ((i',i_1),(j',j_1)) \in P_a \times P_b \end{array} \right\}$$

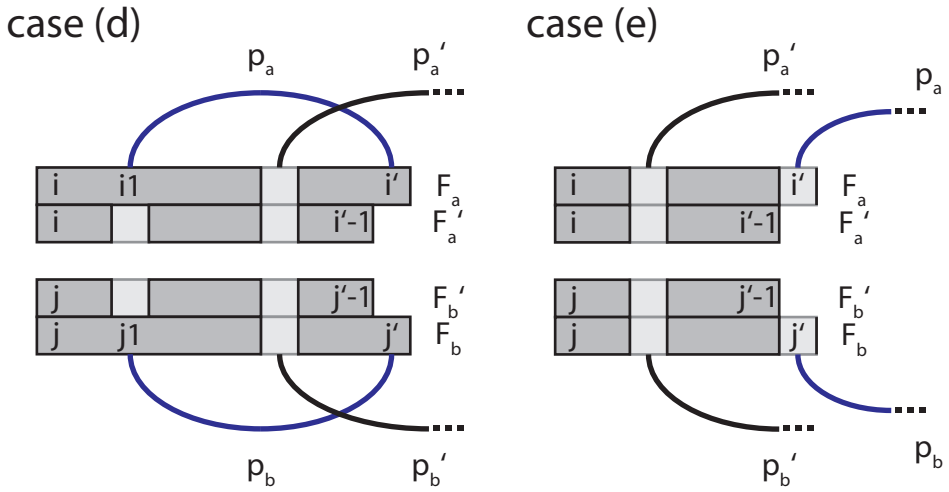


Figure 4.5: Example for cases d) and e) of the Recursion 4.15. The fragment pair $([i,i'],[j,j']) - M$ is visualized as (F_a, F_b) and (F'_a, F'_b) is the fragment pair that the recursion descents to. In case (d), the arc pair (p_a, p_b) is not open in $([i,i'],[j,j'])$ but it is open in its prefix $([i,i'-1],[j,j'-1])$. In case (e) it is the other way around: (p_a, p_b) is open in (F_a, F_b), but no more open in (F'_a, F'_b). The arc pair (p'_a, p'_b) is just some example for other open arc matches and M can contain an arbitrary number of them.

Again we apply the case distinction lemma as follows.

$$C([i,i'],[j,j'] - M) \overset{\text{Cor.1}}{=} C_{AC}([i,i'],[j,j'] - M) \overset{\text{Lem.4}}{=} \min \begin{cases} C_{\mathcal{A}_1}([i,i'],[j,j'] - M) \\ C_{\mathcal{A}_2}([i,i'],[j,j'] - M) \\ C_{\mathcal{A}_3}([i,i'],[j,j'] - M) \\ C_{\mathcal{A}_4}([i,i'],[j,j'] - M) \\ C_{\mathcal{A}_5}([i,i'],[j,j'] - M) \end{cases}$$

The cases for $\mathcal{A}_1$, $\mathcal{A}_2$ and $\mathcal{A}_3$ are proved exactly as in Lemma 7. Note that the sets $\mathcal{A}_4$ and $\mathcal{A}_5$ are empty if and only if the conditions of their respective cases 4.15d and 4.15e are not satisfied. If any of the two sets is not empty, there exists exactly one (i_1, j_1) that satisfies the respective property, since i' and j' are adjacent to at most one arc. In case that $\mathcal{A}_4$ is non empty, for i_1 and j_1 we have

$$\begin{aligned} C_{\mathcal{A}_4}([i,i'],[j,j'] - M) &\overset{\text{Lem.5}}{=} C_{\mathcal{A}_4}([i,i'],[j,j'] - (M \cup \{(p_a,p_b)\})) + \text{arcmatch}(p_a,p_b) \\ &\overset{\text{Lem.6}}{=} C([i,i'],[j,j'] - (M \cup \{(p_a,p_b)\})) + \text{arcmatch}(p_a,p_b) \end{aligned}$$

If $\mathcal{A}_5$ is non empty, $C_{\mathcal{A}_5}([i,i'],[j,j'] - M) = C(([i,i'-1],[j,j'-1]) - (M - \{(p_a,p_b)\}))$ follows directly from $(p_a,p_b) \in M$ which implies $[i,i'],[j,j'] - M = ([i,i'-1],[j,j'-1]) - (M - \{(p_a,p_b)\})$. ∎

As in the previous algorithms the base case for the recursion is

$$C([i, i-1][j, j-1]) := 0 \text{ for all } i, j$$

The optimal cost to align the entire structures (S_a, P_a) and (S_b, P_b) is obtained as the entry $C([1, |S_a|], [1, |S_b|] - \emptyset)$ since the fragment pair covering the entire sequences never has any open arc matches.

Complexity The runtime of a DP algorithm based on the recursion of Lemma 9 is dominated by the number of instances of M that need to be considered. Since M contains sets of arc pairs, the number of instances of M grows exponentially with the number of arc pairs and hence also with the sequence length. More precisely, if k denotes the number of arc matches that can be open for a fragment pair simultaneously, the time and space complexity of the algorithm is $O(nm2^k)$. Hence, the algorithm is fixed-parameter tractable for the parameter k.

Note that the recursion is linear with respect to n and m and not quadratic since only for i' and j' all possible instances need to be considered while i and j remain invariant in the recursion. Therefore, to compute the optimal alignment cost $C([1, |S_a|], [1, |S_b|] - \emptyset)$ only entries of the form $C(([1, i'], [1, j']) - M)$ for $1 \leq i' \leq |S_a|$ and $1 \leq j' \leq |S_b|$ need to be computed.

The extensions of this algorithm described in the following sections all aim at reducing the exponential factor k in the complexity.

4.3.2 Combining Basic Pseudoknot and Nested Alignment

In the previous section we generalized the plain sequence alignment algorithm to handle arbitrary pseudoknots. If the original recursion of the plain sequence alignment algorithm is applied to sequences with a non empty structure each matched arc pair becomes an open arc match for some fragments and we presented a way to cope with these open arc matches. Compared to the plain sequence alignment algorithm the algorithm for nested structures (Lemma 8) can handle nested arc matches and only if it is applied to crossing arcs those may cause open arc matches as indicated by the gray dashed arcs in Figure 4.2(b).

We now present an algorithm that extends the algorithm for nested structures to arbitrary pseudoknots. Basically, all arc matches that the nested algorithm can handle are handled as before and only the few open arc matches that can occur for crossing arcs are handled analogously to the basic pseudoknot algorithm of the previous section (Lemma 9).

Another perspective on that is that both the algorithm for nested structures and the basic pseudoknot alignment algorithm presented in the previous section are extensions of the plain sequence alignment algorithm (Lemma 7). From that perspective, this section presents a combination of both extensions that aims at using a generalized version of the nested recursion for as many arc matches as possible and uses the expensive recursion for crossing arc matches only where necessary. This

results in an improved space and time complexity compared to the basic pseudoknot algorithm since the sets M of open arc matches that make up the exponential factor in the runtime only contain the arc matches that cannot be handled by the recursion for nested structures. In practical examples, it turns out that the number of those arc matches is comparably small.

Partition into Crossing and Non-crossing Arc Pairs

The choice which potential arc matches are handled with the nested recursion and for which the method with exponential complexity must be applied is not unique. Therefore we define the notion of a valid partition of the set of arc pairs $P_a \times P_b$ into a set CR of crossing arc pairs and a set NC of non-crossing arc pairs. The intuition is that all arc matches in NC can be handled with the cheap recursion for nested structures.

Definition 17 (valid partition of arc pairs) Two arc pairs (p_a, p_b) and (p'_a, p'_b) *cross* if and only if $p_a^{\mathrm{L}} < p_a'^{\mathrm{L}} < p_a^{\mathrm{R}} < p_a'^{\mathrm{R}}$ and $p_b^{\mathrm{L}} < p_b'^{\mathrm{L}} < p_b^{\mathrm{R}} < p_b'^{\mathrm{R}}$ or $p_a'^{\mathrm{L}} < p_a^{\mathrm{L}} < p_a'^{\mathrm{R}} < p_a^{\mathrm{R}}$ and $p_b'^{\mathrm{L}} < p_b^{\mathrm{L}} < p_b'^{\mathrm{R}} < p_b^{\mathrm{R}}$. A (bi-)partition of $P_a \times P_b$ into CR and NC is called *valid* if and only if for all (p_a, p_b) and (p'_a, p'_b) in NC it holds that they do not cross. □

Figure 4.6 visualizes arc pairs as rectangles in the plane. If two arc pairs cross, the rectangles partially overlap but note that the converse implication does not hold. In Figure 4.6, for example, the red arc pair (D, I) crosses the dark blue arc pairs like (E, J), but it does not cross the light blue arc pairs like (E, G). Intuitively it is not necessary to consider the latter as crossing since they cannot occur in the same alignment, i.e. there exists no alignment that simultaneously matches D to I and E to G. Another intuition is that there exists no traceback that visits the start and end points of both arc pairs (highlighted as black dots), simultaneously.

For the alignment algorithm a valid partition of $P_a \times P_b$ must be computed in a preprocessing step. This preprocessing step should on the one hand take not too much time and on the other hand result in a partition where CR is as small as possible since the size of CR has a major impact on the runtime and space consumption of the later steps[1].

One practical approach is to lift a valid partition of $P_a \times P_b$ from a partition of the arcs of P_a and P_b by choosing appropriate sets $\mathrm{CR}_a \subseteq P_a$ and $\mathrm{CR}_b \subseteq P_b$ such that $P_a - \mathrm{CR}_a$ and $P_b - \mathrm{CR}_b$ are non-crossing. The partition of $P_a \times P_b$ is then obtained as $\mathrm{CR} = \mathrm{CR}_a \times \mathrm{CR}_b$ and $\mathrm{NC} = P_a \times P_b - \mathrm{CR}$. However, this does not work for arbitrary non-crossing sets $P_a - \mathrm{CR}_a$ and $P_b - \mathrm{CR}_b$. For example, in Figure 4.6 choosing $\mathrm{CR}_a = \{A, B, E\}$ and $\mathrm{CR}_b = \{I\}$ is not valid, since $\{A, B, E\} \times \{I\}$ contains none of the two crossing arc pairs (A, G) and (D, I). Valid partitions are obtained,

[1] Actually not only the number of arc pairs in CR plays a role for the runtime, but also the length of each arc and the way in which they overlap. In that sense, minimizing just the size of CR is a heuristic but finding the partition for which the alignment algorithm runs fastest is a complex optimization problem that might be as complex as solving the alignment problem itself.

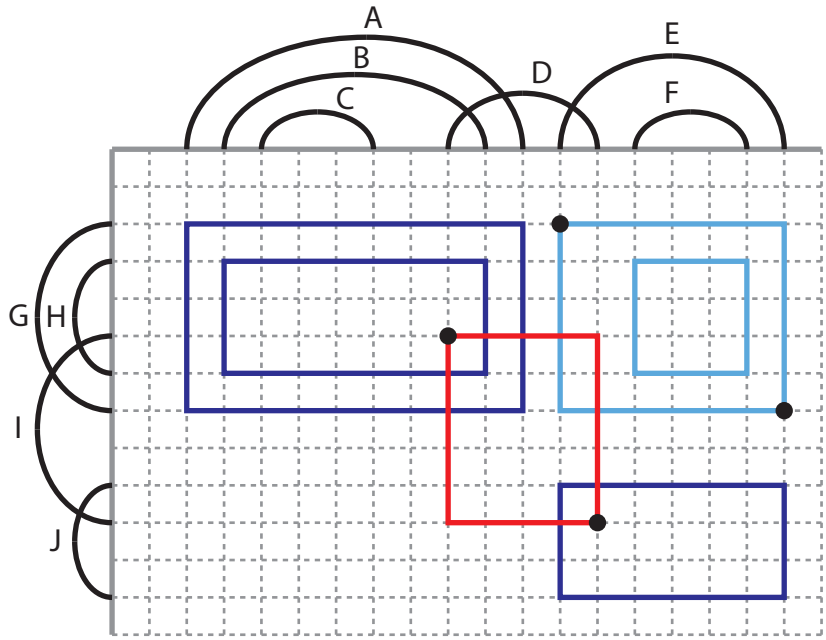


Figure 4.6: Visualization of the arc pairs of two sequences as rectangles. The first sequence is drawn horizontally (left to right) and has arcs A to F, the second sequence is drawn vertically (top down) and has arcs G to J. To maintain readability, only some arcs pairs are visualized. Among the visualized arc pairs, it would be sufficient to place the red one in the crossing set CR to obtain a valid partition. The red arc pair crosses with the dark blue arc pairs but not with the light blue ones since their start and end points (highlighted as black dots) cannot be aligned simultaneously.

for example, if CR_a and CR_b both contain all left crossing arcs or both contain all right crossing arcs.

Definition 18 (left/right crossing) For two crossing arcs p and p' with $p^{\mathrm{L}} < p'^{\mathrm{L}} < p^{\mathrm{R}} < p'^{\mathrm{R}}$, we say that p is right crossing and p' is left crossing. □

The following lemma gives a sufficient criterion for a valid partition that can easily be constructed.

Lemma 10 (left/right criterion for a partition) *For*

$$CR = \{\, p_a \in P_a \,|\, p_a \text{ is left crossing} \,\} \times \{\, p_b \in P_b \,|\, p_b \text{ is left crossing} \,\}$$

or

$$CR = \{\, p_a \in P_a \,|\, p_a \text{ is right crossing} \,\} \times \{\, p_b \in P_b \,|\, p_b \text{ is right crossing} \,\}$$

the partition of $P_a \times P_b$ into CR and $NC = P_a \times P_b - CR$ is valid. □

Proof Let (p_a, p_b) and (p'_a, p'_b) be two crossing arc pairs. We show that at least one of (p_a, p_b) and (p'_a, p'_b) is contained in CR. Assume w.l.o.g. $p_a^{\mathrm{L}} < p_a'^{\mathrm{L}} < p_a^{\mathrm{R}} < p_a'^{\mathrm{R}}$ and $p_b^{\mathrm{L}} < p_b'^{\mathrm{L}} < p_b^{\mathrm{R}} < p_b'^{\mathrm{R}}$. Then p_a and p_b are right crossing and p'_a and p'_b are left crossing. Hence (p_a, p_b) is contained in CR if it contains all pairs of right crossing arcs and (p'_a, p'_b) is contained in CR if it contains all pairs of left crossing arcs. ■

A partition according to Lemma 10 is not necessarily minimal in the sense that CR contains as few elements as possible. For the structures in Figure 4.6, for example, the right and left crossing criteria would yield $\mathrm{CR} = \{A, B, C, D\} \times \{G, H, I\}$ and $\mathrm{CR} = \{D, E, F\} \times \{I, J\}$, respectively. While for those $|\mathrm{CR}| = 12$ and $|\mathrm{CR}| = 6$, respectively, there exists also a partition with $|\mathrm{CR}| = 3$, namely for $\mathrm{CR} = \{(D, I), (D, J), (E, I)\}$.

Nevertheless, for sequences with few and small pseudoknots also with the left or right crossing criterion the set CR is small and it can easily be constructed in linear time with respect to sequence length. In an implementation it is not even necessary to explicitly construct the set, but it suffices to store for each arc a flag whether it is left or right crossing. Then, whenever it needs to be checked whether an arc match is contained in CR or not, only the respective flags of the two arcs need to be checked.

From now on, we just assume to have a valid partition where CR is reasonably small. This partition can be obtained according to Lemma 10 but a more involved preprocessing step to find even better partitions might speed up the algorithm even further.

The Algorithm Recursion

The recursion of the algorithm is given in the following lemma.

Lemma 11 *For a valid partition of $P_a \times P_b$ into CR and NC, any $M \subset$ CR and i, i', j, i' such that $1 \leq i < i' \leq |S_a|$, $1 \leq j < j' \leq |S_b|$ and such that $([i, i'], [j, j']) - M$ is valid.*

$$C(([i,i'][j,j']) - M) = \min \tag{4.16}$$

$$\begin{cases} C(([i,i'-1],[j,j']) - M) + \text{gap}_1(i') & (4.16a) \\ C(([i,i'],[j,j'-1]) - M) + \text{gap}_2(j') & (4.16b) \\ C(([i,i'-1],[j,j'-1]) - M) + \text{basematch}(i',j') & (4.16c) \\ \textit{if } \exists (p_a,p_b) = ((i_1,i'),(j_1,j')) \in NC & (4.16d) \\ \quad \min \left\{ \begin{array}{l|l} C(([i,i_1-1],[j,j_1-1]) - M_1) + & \\ C(([i_1+1,i'-1][j_1+1,j'-1]) - M_2) + & M_3 = M_1 \cap M_2 \\ \sum_{am \in M_3 \cup \{(p_a,p_b)\}} \text{arcmatch}(am) & M = (M_1 \cup M_2) - M_3 \end{array} \right\} & \\ \textit{if } \exists (p_a,p_b) = ((i_1,i'),(j_1,j')) \in CR \textit{ for some } i_1 \geq i,\ j_1 \geq j & (4.16e) \\ \quad C(([i,i'-1],[j,j'-1]) - (M \cup \{(p_a,p_b)\}) + \text{arcmatch}(p_a,p_b) & \\ \textit{if } \exists (p_a,p_b) \in M \textit{ with } p_a^L = i', p_b^L = j' \textit{ or } p_a^R = i', p_b^R = j' & (4.16f) \\ \quad C(([i,i'-1],[j,j'-1]) - (M - \{(p_a,p_b)\})) & \end{cases}$$

Again, cases referring to invalid items are implicitly skipped. □

In case (4.16d) the recursion applies an extended variant of the recursion for nested structures for arc matches in NC. Arc matches in CR are handled as in the basic pseudoknot algorithm in cases (4.16e) and (4.16f). The recursion case (4.16d) differs from the nested recursion case (4.6d) in order to handle arcs pairs from CR that cross the arc match $(p_a, p_b) \in$ NC. Therefore it minimizes over all possible sets M_3 of arc matches in CR that cross the currently considered arc match $(p_a, p_b) \in$ NC. M_3 and M together uniquely determine the sets M_1 and M_2 of open arc matches for the two fragment pairs $([i, i_1 - 1], [j, j_1 - 1])$ and $([i_1 + 1, i' - 1], [j_1 + 1, j' - 1])$ that the recursion descents to. A visualization of the recursion case (4.16d) is given in Figure 4.7.

Proof The case distinction corresponds to the following partition of the set of alignments $\mathcal{A}$ that align $([i, i'], [j, j']) - M$ and contain no open arc matches for this fragment pair:

$$\mathcal{A}_1 := \left\{ A \in \mathcal{A} \,\middle|\, (i', -) \in A \right\}$$
$$\mathcal{A}_2 := \left\{ A \in \mathcal{A} \,\middle|\, (-, j') \in A \right\}$$
$$\mathcal{A}_3 := \left\{ A \in \mathcal{A} \,\middle|\, \begin{array}{l} (i',j') \in A \wedge \nexists (i_1,j_1) \in A. \\ \quad ((i_1,i'),(j_1,j')) \in P_a \times P_b \vee ((i',i_1),(j',j_1)) \in P_a \times P_b \end{array} \right\}$$
$$\mathcal{A}_4 := \left\{ A \in \mathcal{A} \,\middle|\, \begin{array}{l} (i',j') \in A \wedge \exists (i_1,j_1) \in A. \\ \quad ((i_1,i'),(j_1,j')) \in \text{NC} \wedge i_1 \geq i \wedge j_1 \geq j \end{array} \right\}$$

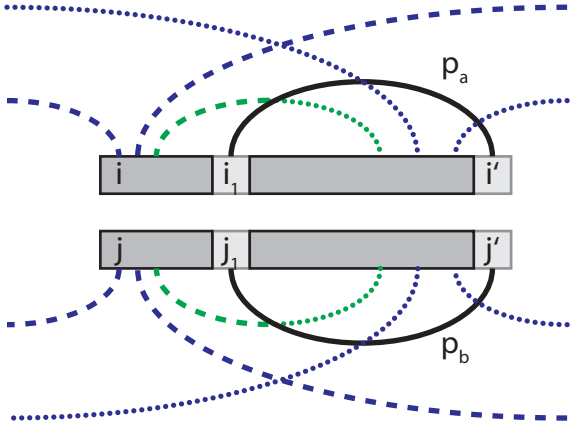


Figure 4.7: Visualization of recursion case (4.16d). The dashed and dotted arc pairs represent the sets M, M_1, M_2, and M_3 as follows. Each of those drawn arcs pairs represents a set of possible arc pairs, i.e. more than one arc pair of each shown type can occur. M consists of all blue arc pairs and M_3 of the green ones. From both colors M_1 contains the dashed arc pairs and M_2 the dotted ones. Effectively the recursion minimizes over all possible instances of M_3; since M is fixed for each matrix entry this uniquely determines M_1 and M_2. Since the black arc pair (p_a, p_b) is contained in NC, all blue and green arc pairs must be contained in CR.

$$\mathcal{A}_5 := \left\{ A \in \mathcal{A} \,\middle|\, \begin{array}{l} (i',j') \in A \wedge \exists (i_1,j_1) \in A. \\ \quad ((i_1,i'),(j_1,j')) \in \mathrm{CR} \wedge i_1 \geq i \wedge j_1 \geq j \end{array} \right\}$$

$$\mathcal{A}_6 := \left\{ A \in \mathcal{A} \,\middle|\, \begin{array}{ll} (i',j') \in A \wedge \exists (i_1,j_1) \in A. & \\ \quad ((i_1,i'),(j_1,j')) \in \mathrm{CR} \wedge i_1 < i \wedge j_1 < j & \vee \\ \quad ((i',i_1),(j',j_1)) \in \mathrm{CR} & \end{array} \right\}$$

Note that the case distinction is exhaustive: there does not exist any (i_1, j_1) with $((i', i_1), (j', j_1)) \in \mathrm{NC}$, or with $((i_1, i'), (j_1, j')) \in \mathrm{NC}$ and $i_1 < i$, $j_1 < j$ since the fragment pair $([i, i'][j, j']) - M$ contains no open arc matches and M is a subset of CR. Note that the invariant that $M \subseteq \mathrm{CR}$ is maintained through the recursion, i.e. if $M \subseteq \mathrm{CR}$ then this also holds for the modified values of M that the recursion descents to. The only non-trivial case for this is case (4.16d). In this case all elements of M_1 and M_2 are either contained in M or in M_3. $M \subseteq \mathrm{CR}$ holds by induction hypothesis and $M_3 \subseteq \mathrm{CR}$ follows since all elements of M_3 cross the arc match (p_a, p_b) which is contained in NC (see the green arc pair of Figure 4.7).

The remainder of the proof is analogous to the proof of the basic pseudoknot algorithm (Lemma 9), only for the recursive case 4.16d corresponding to $\mathcal{A}_4$ some new arguments are necessary. This case is similar as in Lemma 8, but we have to make a further case distinction over the possible instances of M_3, which capture the arc matches connecting the part before and the part below the arc match

$((i_1, i'), (j_1, j'))$. While these do not exist in nested structures, they can occur as elements of CR in this combined algorithm (see the green arc pair of Figure 4.7).

To show the correctness of case (4.16d) we partition $\mathcal{A}_4$ into $\bigcup_{M_3} \mathcal{A}_4^{M_3}$, where each $\mathcal{A}_4^{M_3}$ is the set of alignments $A \in \mathcal{A}_4$ for which M_3 is the set of arc matches am of A which satisfy $am \in ([i, i_1 - 1] \times [i_1 + 1, i' - 1]) \times ([j, j_1 - 1] \times [j_1 + 1, j' - 1])$. Then

$$C_{\mathcal{A}_4}(([i, i'][j, j']) - M) \overset{\text{Lem.4}}{=} \min_{\mathcal{A}_4^{M_3}} C_{\mathcal{A}_4^{M_3}}(([i, i'][j, j']) - M) \tag{4.17}$$

For each M_3 the sets M_1 and M_2 are uniquely determined: M_1 contains the arc pairs from $M \cup M_3$ with one endpoint within $[i, i_1 - 1]\times, [j, j_1 - 1]$, and M_2 contains the ones with one endpoint within $([i_1 + 1, i' - 1] \times [j_1 + 1, j' - 1]$. For these M_1 and M_2 we have

$$\begin{aligned}
C_{\mathcal{A}_4^{M_3}}([i, i'][j, j'] - M) &\overset{\text{Lem.5}}{=} C_{\mathcal{A}_4^{M_3}}(([i, i_1 - 1], [j, j_1 - 1]) - M_1) \\
&\quad + C_{\mathcal{A}_4^{M_3}}(([i_1 + 1, i' - 1][j_1 + 1, j' - 1]) - M_2) \qquad (4.18) \\
&\quad + \sum_{(p'_a, p'_b) \in M_3 \cup \{(p_a, p_b)\}} C_{\mathcal{A}_4^{M_3}}(\{[p'^{\mathrm{L}}_a], [p'^{\mathrm{R}}_a]\}, \{[p'^{\mathrm{L}}_b], [p'^{\mathrm{R}}_b]\}) \\
&\overset{\text{Def.13}}{=} C_{\mathcal{A}_4^{M_3}}(([i, i_1 - 1], [j, j_1 - 1]) - M_1) \\
&\quad + C_{\mathcal{A}_4^{M_3}}(([i_1 + 1, i' - 1][j_1 + 1, j' - 1]) - M_2) \qquad (4.19) \\
&\quad + \sum_{am \in M_3 \cup \{(p_a, p_b)\}} \mathrm{arcmatch}(am) \\
&\overset{\text{Lem.6}}{=} C(([i, i_1 - 1], [j, j_1 - 1]) - M_1) \\
&\quad + C(([i_1 + 1, i' - 1][j_1 + 1, j' - 1]) - M_2) \qquad (4.20) \\
&\quad + \sum_{am \in M_3 \cup \{(p_a, p_b)\}} \mathrm{arcmatch}(am)
\end{aligned}$$

■

Complexity

Items $C(([i, i'][j, j']) - M)$ are computed for $1 \le i \le i' \le n$, $1 \le j \le j' \le m$ for all possible sets of open arc matches $M \subseteq \mathrm{CR}$. Hence, the total $O(n^2 m^2 2^k)$ items are computed , where k is the number of arc matches in CR that can be open in a fragment pair simultaneously. The same method that is used to reduce the space complexity of the algorithm for nested structures from $O(n^2 m^2)$ to $O(nm)$ [27] can also be applied in this combined algorithm. Hence the space complexity is in $O(nm2^k)$.

All recursion cases except for (4.16d) need only constant time. This case minimizes over all possible instances of M_3 of which there are at most 2^k many. Hence the

computation requires $O(n^2m^2 2^k \cdot 2^k) = O(n^2m^2 2^{2k})$ time. Effectively, compared to the basic pseudoknot alignment algorithm of the previous section (Lemma 9) this algorithm reduces the exponential parameter k from the number of open arc pairs to the number of crossing open arc pairs.

4.3.3 The Final Algorithm with Stem Optimization

The exponential parameter of the algorithm can be further reduced by considering entire stems instead of single arcs. This idea takes advantage of the fact that arcs in naturally occurring structures usually group into stems such that there are much less crossing stems than crossing arcs.

Lifting Concepts from Arcs to Stems

Definition 19 (stem) For an arc annotated sequence (S, P), a stem $Q \subseteq P$ is a set of arcs $\{p_1, \ldots, p_k\} \subseteq P$ with $p_1^{\mathrm{L}} < \cdots < p_k^{\mathrm{L}} < p_k^{\mathrm{R}} < \cdots < p_1^{\mathrm{R}}$ such that no end of arcs in $P - Q$ is in one of the intervals $[p_1^{\mathrm{L}}..p_k^{\mathrm{L}}]$ or $[p_k^{\mathrm{R}}..p_1^{\mathrm{R}}]$. □

Intuitively, a stem is a set of arcs that are stacked on top of each other without any other arcs that interleave this structure. Note that a stem is not required to be maximal: given a stem $\{p_1, \ldots, p_k\} \subseteq P$ with $p_1^{\mathrm{L}} < \cdots < p_k^{\mathrm{L}} < p_k^{\mathrm{R}} < \cdots < p_1^{\mathrm{R}}$, for all $1 \leq i \leq j \leq k$ the set $\{p_i, \ldots, p_j\}$ is also a stem.

The stem optimization can be best thought of as considering entire stems of arcs as just one 'big' arc. The algorithm does not match pairs of arcs but pairs of stems. Therefore, we now lift the notion of an arc pair to stems. A stem pair is characterized by its innermost and outermost arc pair.

Definition 20 (stem pair) The *stem pair* of two stems $Q_a \subseteq P_a$ and $Q_b \subseteq P_b$ is characterized by the pair (a_O, a_I) of arc pairs, where $a_O = (p_{Oa}, p_{Ob})$ is the pair of the outermost arcs and $a_I = (p_{Ia}, p_{Ib})$ is the pair of the innermost arcs of Q_a and Q_b, i.e. Q_k consists of the arcs $P_k \cap [p_{Ok}^{\mathrm{L}}..p_{Ik}^{\mathrm{L}}] \times [p_{Ik}^{\mathrm{R}}..p_{Ok}^{\mathrm{R}}]$ for $k \in \{a, b\}$.

The stem pair is *matched by an alignment* A, if both a_O and a_I are matched by A. For a given partition of $P_1 \times P_2$ into CR and NC, we denote with SP_{CR} the set of stem pairs (a_O, a_I) with $\{a_O, a_I\} \subseteq \mathrm{CR}$. □

An example for a stem pair is given in Figure 4.8. Note that if a stem pair is matched by an alignment, only the inner- and outermost arc pairs are required to be matched and nothing is said about whether and how the arcs in between are matched. Consequently, whereas the algorithm of the previous section has one separate case for each possibility how to match the other arcs within the two stems, all of those cases can now be captured in just one case. This is the central point why the stem optimization leads to a speed-up.

Also note that if any possible arc pair within a stem pair is contained in CR, it is reasonable to assume that all possible arc pairs within that stem pair are contained in CR. If CR is constructed according to the left/right crossing criterion (Lemma 10)

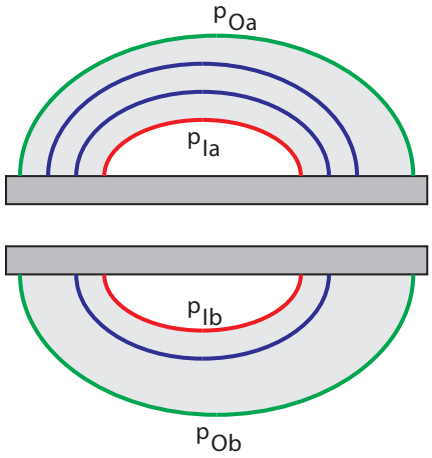


Figure 4.8: A stem pair characterized by the pair of outermost arcs $a_O = (p_{Oa}, p_{Ob})$ shown in green and the pair of innermost arcs $a_I = (p_{Ia}, p_{Ib})$ shown in red.

this holds since if any arc within a stem is left/right crossing, all others are left/right crossing, too. Furthermore, if we instead assume CR to be minimal either in the sense that it has minimal cardinality or in the weaker sense that removing any element from CR would result in an invalid partition the assumption would still hold: as soon as one arc pair is contained in NC, all other arc pairs within the same stem pair can also safely be added to NC, since all arc pairs within a stem pair cross the same arc pairs. We now generalize Definition 16 from arc pairs to stem pairs.

Definition 21 (subtracting stem pairs from fragment pairs) Let (F_a, F_b) be a fragment pair and SP a set of stem pairs such that

I $\forall((p_{Oa}, p_{Ob}), (p_{Ia}, p_{Ib})) \in SP.$
$p^{\mathrm{L}}_{Oa} \in \hat{F}_a \Leftrightarrow p^{\mathrm{L}}_{Ia} \in \hat{F}_a \Leftrightarrow p^{\mathrm{L}}_{Ob} \in \hat{F}_b \Leftrightarrow p^{\mathrm{L}}_{Ib} \in \hat{F}_b \Leftrightarrow$
$p^{\mathrm{R}}_{Oa} \notin \hat{F}_a \Leftrightarrow p^{\mathrm{R}}_{Ia} \notin \hat{F}_a \Leftrightarrow p^{\mathrm{R}}_{Ob} \notin \hat{F}_b \Leftrightarrow p^{\mathrm{R}}_{Ib} \notin \hat{F}_b$

II there exists an alignment A such that all stem pairs of SP are matched by A.

III No two stem pairs in SP can be replaced by one larger stem pair, i.e. for all distinct $((p_{Oa}, p_{Ob}), (p_{Ia}, p_{Ib})) \in SP$ and $((p'_{Oa}, p'_{Ob}), (p'_{Ia}, p'_{Ib})) \in SP$ it holds that $((p_{Oa}, p_{Ob}), (p_{Ia}, p_{Ib}))$ is not a stem pair.

Then we denote with $(F_a, F_b) - SP$ the fragment pair (F'_a, F'_b) where F'_a and F'_b are

the fragments of minimal degree satisfying

$$\hat{F}'_a = \left\{ x \in \hat{F}_a \,\middle|\, \forall((p_{Oa}, p_{Ob}), (p_{Ia}, p_{Ib})) \in SP.\ x \notin [p^{\mathrm{L}}_{Oa}, p^{\mathrm{L}}_{Ia}] \cup [p^{\mathrm{R}}_{Ia}, p^{\mathrm{R}}_{Oa}] \right\} \tag{4.21}$$

$$\hat{F}'_b = \left\{ y \in \hat{F}_b \,\middle|\, \forall((p_{Oa}, p_{Ob}), (p_{Ia}, p_{Ib})) \in SP.\ y \notin [p^{\mathrm{L}}_{Ob}, p^{\mathrm{L}}_{Ib}] \cup [p^{\mathrm{R}}_{Ib}, p^{\mathrm{R}}_{Ob}] \right\} \tag{4.22}$$

$$\forall((p_{Oa}, p_{Ob}), (p_{Ia}, p_{Ib})) \in SP.\ \forall(x, y) \in \{(p^{\mathrm{L}}_{Oa}, p^{\mathrm{L}}_{Ob}), (p^{\mathrm{R}}_{Oa}, p^{\mathrm{R}}_{Ob})\}.\ \forall i. \tag{4.23}$$
$$F'_a[i]^{\mathrm{R}} < x < F'_a[i+1]^{\mathrm{L}} \Leftrightarrow F'_b[i]^{\mathrm{R}} < y < F'_b[i+1]^{\mathrm{L}}$$

If $(F_a, F_b) - SP$ does not satisfy restriction I, II or III we call it invalid otherwise valid. □

The only aspect of the definition that is not directly lifted from Definition 16 is the additional restriction III. This restriction is necessary to ensure that the set SP is always as small as possible and that no redundant possibilities for SP are considered.

Precomputing the Alignment of Stem Pairs

Whereas the cost to match an arc pair (p_a, p_b) is directly given as $\mathrm{arcmatch}(p_a, p_b)$, the cost to match two stems cannot be determined in constant time. Therefore, the alignment costs for all possible stem pairs are determined by dynamic programming in a preprocessing step. The computation is similar to a basic sequence structure alignment without branching.

The optimal cost to align a stem pair $sp = ((p_{Oa}, p_{Ob}), (p_{Ia}, p_{Ib}))$ is defined as

$$\mathrm{stemmatch}(sp) := C(\{[p^{\mathrm{L}}_{Oa}, p^{\mathrm{L}}_{Ia}], [, p^{\mathrm{R}}_{Ia}, p^{\mathrm{R}}_{Oa}]\}, \{[p^{\mathrm{L}}_{Ob}, p^{\mathrm{L}}_{Ib}], [, p^{\mathrm{R}}_{Ib}, p^{\mathrm{R}}_{Ob}]\})$$

and can be computed according to the following lemma.

Lemma 12 *For any stem pair $sp = ((p_{Oa}, p_{Ob}), (p_{Ia}, p_{Ib}))$ and i, i', j, j' such that $p^L_{Oa} \le i \le p^L_{Ia}$, $p^R_{Ia} \le i' \le p^R_{Oa}$, $p^L_{Ob} \le j \le p^L_{Ib}$, $p^R_{Ib} \le j' \le p^R_{Ob}$, is holds*

$$C(\{[i, p^L_{I_a}], [p^R_{I_a}, i']\}, \{[j, {p_{I_b}}^L], [p^R_{I_b}, j']\}) = \min \tag{4.24}$$
$$\begin{cases}
C(\{[i{+}1, p^L_{I_a}], [p^R_{I_a}, i' \quad]\}, \{[j \quad, p^L_{I_b}], [p^R_{I_b}, j' \quad]\}) + \mathrm{gap}_a(i) & \text{(4.24a)}\\
C(\{[i \quad, p^L_{I_a}], [p^R_{I_a}, i'{-}1]\}, \{[j \quad, p^L_{I_b}], [p^R_{I_b}, j' \quad]\}) + \mathrm{gap}_a(i') & \text{(4.24b)}\\
C(\{[i \quad, p^L_{I_a}], [p^R_{I_a}, i' \quad]\}, \{[j{+}1, p^L_{I_b}], [p^R_{I_b}, j' \quad]\}) + \mathrm{gap}_b(j) & \text{(4.24c)}\\
C(\{[i \quad, p^L_{I_a}], [p^R_{I_a}, i' \quad]\}, \{[j \quad, p^L_{I_b}], [p^R_{I_b}, j'{-}1]\}) + \mathrm{gap}_b(j') & \text{(4.24d)}\\
C(\{[i{+}1, p^L_{I_a}], [p^R_{I_a}, i' \quad]\}, \{[j{+}1, p^L_{I_b}], [p^R_{I_b}, j' \quad]\}) + \mathrm{basematch}(i, j) & \text{(4.24e)}\\
C(\{[i \quad, p^L_{I_a}], [p^R_{I_a}, i'{-}1]\}, \{[j \quad, p^L_{I_b}], [{p_{I_b}}^R, j'{-}1]\}) + \mathrm{basematch}(i', j') & \text{(4.24f)}\\
\textit{if } (i, i') \in P_a \textit{ and } (j, j') \in P_b & \text{(4.24g)}\\
\quad C(\{[i{+}1, p^L_{I_a}], [p^R_{I_a}, i'{-}1]\}, \{[j{+}1, p^L_{I_b}], [p^R_{I_b}, j'{-}1]\}) + \mathrm{arcmatch}((i, i'), (j, j'))
\end{cases}$$

In the recursion the cases where any of i, i', j, j' lies outside the specified range are implicitly skipped. □

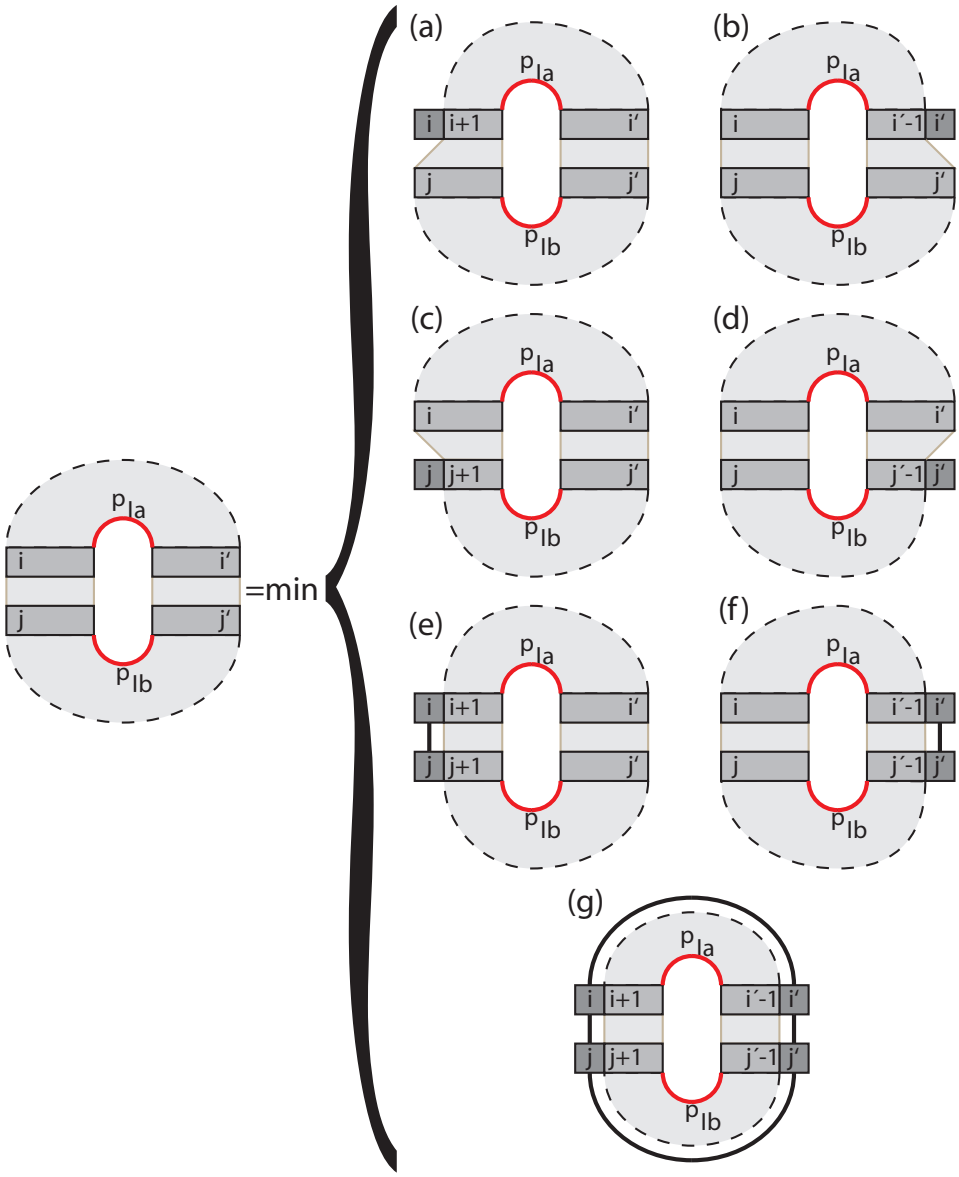


Figure 4.9: Visualization of the recursion given in Lemma 12.

PROOF As usual, the correctness can by shown by first applying the case distinction lemma (Lemma 4) and then applying for each case the independence and cost locality lemmata (Lemma 5 and Lemma 6) to separate the fixed part of the alignment (i.e. the gap, base match, or arc match of the respective case) from the remaining part. Since the proof is completely analogous to the previous ones details are omitted here. Just note that the case distinction into cases 4.24a to 4.24g is exhaustive. Intuitively, each alignment of two stems can be constructed by starting with the innermost two arcs and then extending this step by step with gaps, base matches and arc matches. ■

The recursion of Lemma 12 is visualized in Figure 4.9. The recursion is combined with the base case

$$C(\{[p_a^{\mathrm{L}}], [p_a^{\mathrm{R}}]\}, \{[p_b^{\mathrm{L}}], [p_b^{\mathrm{R}}]\}) = \mathrm{arcmatch}(p_a, p_b)$$

for all $(p_a, p_b) \in P_a \times P_b$. Formally this is not correct since by the definition of C the arcs p_a and p_b are not matched necessarily. However in the main recursion of the algorithm we use the computed values only in a context where this can safely be assumed since situations where they are not matched are covered by other cases. In principle the algorithm can also be implemented with the base cases

$$C(\{[p_a^{\mathrm{L}} + 1, p_a^{\mathrm{L}}], [p_a^{\mathrm{R}}, p_a^{\mathrm{R}} - 1]\}, \{[p_b^{\mathrm{L}} + 1, p_b^{\mathrm{L}}], [p_b^{\mathrm{R}}, p_b^{\mathrm{R}} - 1]\}) = 0$$

which lets the recursion terminate one step later but includes the cases where p_a and p_b are not matched.

The Main Recursion

In order to present the main recursion of the algorithm the following additional syntax is introduced.

Definition 22 (start and end point of arc matches) For any arc pair we denote its start and endpoint as $\nwarrow(p_a, p_b) := (p_a^{\mathrm{L}}, p_b^{\mathrm{L}})$ and $\searrow(p_a, p_b) := (p_a^{\mathrm{R}}, p_b^{\mathrm{R}})$, respectively. □

The recursion is an extension of the recursion of the previous algorithm (Lemma 11). The main change is that instead of considering each arc pair in CR separately, entire stems of arcs are considered in one step simultaneously. The set of open arc matches M of the previous algorithm is therefore replaced by a set of open stem pairs SP. The recursion is given in the following lemma and visualized in Figure 4.10.

Lemma 13 *For a valid partition of $P_a \times P_b$ into CR and NC, any $SP \subseteq SP_{CR}$ and i, i', j, i' such that $1 \leq i < i' \leq |S_a|$, $1 \leq j < j' \leq |S_b|$ and such that $([i, i'], [j, j']) - SP$*

is valid

$$C(([i,i'][j,j']) - SP) = \min \tag{4.25}$$

$$\begin{cases}
C(([i,i'-1],[j,j']) - SP) + \text{gap}_1(i') & (4.25a)\\
C(([i,i'],[j,j'-1]) - SP) + \text{gap}_2(j') & (4.25b)\\
C(([i,i'-1],[j,j'-1]) - SP) + \text{basematch}(i',j') & (4.25c)\\
\textit{if } \exists (p_a,p_b) = ((i_1,i'),(j_1,j')) \in NC & (4.25d)\\
\quad \min \left\{ \left. \begin{array}{l} C(([i,i_1-1],[j,j_1-1]) - SP_1) + \\ C(([i_1+1,i'-1][j_1+1,j'-1]) - SP_2) \\ \quad + \sum\limits_{(a_o,a_I)\in SP_3} \text{stemmatch}(a_O,a_I) \\ \quad + \text{arcmatch}(p_a,p_b) \end{array} \right| \begin{array}{l} SP_3 = SP_1 \cap SP_2 \\ SP = (SP_1 \cup SP_2) - SP_3 \end{array} \right\} & \\
\textit{if } \exists (a_O,a_I) \in SP_{CR} - SP \textit{ with } \searrow(a_O) = (i',j') & (4.25e)\\
\quad \min \left\{ \left. \begin{array}{l} C(([i,i_1-1],[j,j_1-1]) - (SP \cup \{(a_o,a_I)\})) \\ \quad + \text{stemmatch}(a_o,a_I) \end{array} \right| \begin{array}{l} (a_O,a_I) \in SP_{CR}, \textit{ where} \\ \searrow a_O = (i',j') \textit{ and} \\ \searrow a_I = (i_1,j_1) \end{array} \right\} & \\
\textit{if } \exists (a_O,a_I) \in SP \textit{ and some } (i_1,j_1) \textit{ with} & (4.25f)\\
\quad \nwarrow a_O = (i_1,j_1) \wedge \nwarrow a_I = (i',j') \textit{ or } \searrow a_I = (i_1,j_1) \wedge \searrow a_O = (i',j') & \\
\quad C(([i,i_1-1],[j,j_1-1]) - (SP - \{(a_O,a_I)\})) &
\end{cases}$$

Again, cases referring to invalid items are implicitly skipped. □

Proof The proof is analogous to the proof of Lemma 11 and the case distinction over the set of possible alignments $\mathcal{A}$ that align $([i,i'],[j,j']) - SP$ is exactly the same:

$$\mathcal{A}_1 := \left\{ A \in \mathcal{A} \middle| (i',-) \in A \right\}$$
$$\mathcal{A}_2 := \left\{ A \in \mathcal{A} \middle| (-,j') \in A \right\}$$
$$\mathcal{A}_3 := \left\{ A \in \mathcal{A} \middle| \begin{array}{l} (i',j') \in A \wedge \nexists (i_1,j_1) \in A. \\ \quad ((i_1,i'),(j_1,j')) \in P_a \times P_b \quad \vee ((i',i_1),(j',j_1)) \in P_a \times P_b \end{array} \right\}$$
$$\mathcal{A}_4 := \left\{ A \in \mathcal{A} \middle| (i',j') \in A \wedge \exists (i_1,j_1) \in A.\ ((i_1,i'),(j_1,j')) \in \text{NC} \wedge i_1 \geq i \wedge j_1 \geq j \right\}$$
$$\mathcal{A}_5 := \left\{ A \in \mathcal{A} \middle| (i',j') \in A \wedge \exists (i_0,j_0) \in A.\ ((i_0,i'),(j_0,j')) \in \text{CR} \wedge i_0 \geq i \wedge j_0 \geq j \right\}$$
$$\mathcal{A}_6 := \left\{ A \in \mathcal{A} \middle| \begin{array}{l} (i',j') \in A \wedge \exists (i_0,j_0) \in A. \\ \quad ((i_0,i'),(j_0,j')) \in \text{CR} \wedge i_0 < i \wedge j_0 < j \vee ((i',i_0),(j',j_0)) \in \text{CR} \end{array} \right\}$$

After applying the case distinction lemma (Lemma 4) as usual, the cases for $\mathcal{A}_1$, $\mathcal{A}_2$ and $\mathcal{A}_3$ are analogous to the previous algorithms. It remains to show that

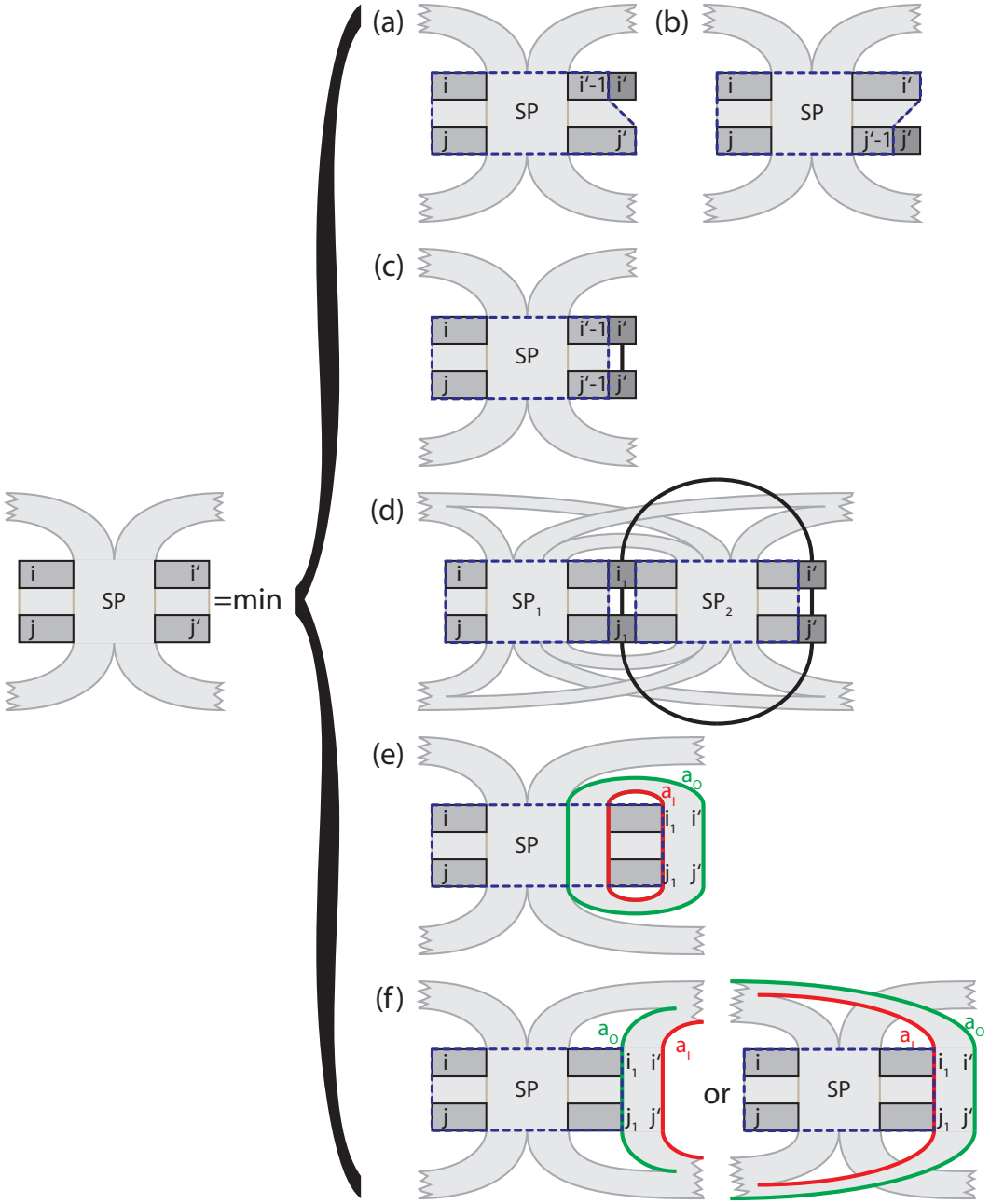


Figure 4.10: Visualization of the recursion given in Lemma 13. The dashed blue boxes mark the parts to which the algorithm recursively descents. The open stem pairs shown in light gray are visualized as one big gap but may also cause several gaps that are distributed over the fragment pair. Furthermore the open stem pairs can exist to the left, to the right, to both directions simultaneously (as visualized), or, for $SP = \emptyset$, not at all.

$C_{\mathcal{A}_4}(([i,i'],[j,j'])-SP)$, $C_{\mathcal{A}_5}(([i,i'],[j,j'])-SP)$, and $C_{\mathcal{A}_6}(([i,i'],[j,j'])-SP)$ equal the values computed in cases (4.25d), (4.25e) and (4.25f), respectively. For all of them we assume that the respective sets $\mathcal{A}_4$, $\mathcal{A}_5$, and $\mathcal{A}_6$ are not empty; if they are empty the conditions of the respective cases are not satisfied and hence they do not occur in the recursion. If the sets are not empty, the values i_0, j_0, i_1, and j_0 within the definition of $\mathcal{A}_4$, $\mathcal{A}_5$, and $\mathcal{A}_6$ are equal for all alignments A, since the sets of arcs NC and CR are fixed and there can be at most one arc adjacent to i' and j', respectively. Therefore, we assume some fixed the values for i_0, j_0, i_1, and j_1 when we look at each of the cases now separately.

We start with case (4.25d). Similar to the proof of Lemma 11 we partition $\mathcal{A}_4$ into $\bigcup_{SP_3} \mathcal{A}_4^{SP_3}$ where each $\mathcal{A}_4^{SP_3}$ is the set of alignments $A \in \mathcal{A}_4$ for which SP_3 is the set of stem pairs that are matched by A, are not contained in any larger stem pair matched by A and have one end in the fragment pair $([i,i_1-1],[j,j_1-1])$ and the other end in the fragment pair $([i_1+1,i'-1],[j_1+1,j'-1])$. The restriction to such maximal stem pairs is valid, since by restriction III of Definition 21 the algorithm only considers those. On the partition the case distinction lemma is applied as usual:

$$C_{\mathcal{A}_4}(([i,i'][j,j'])-SP) \overset{\text{Lem.4}}{=} \min_{\mathcal{A}_4^{SP_3}} C_{\mathcal{A}_4^{SP_3}}(([i,i'][j,j'])-SP) \tag{4.26}$$

Consider now a some fixed SP_3. Together with SP this also fixes some uniquely determined SP_1 and SP_2 that satisfy the restrictions stated in (4.25d). For these we have

$$\begin{aligned} C_{\mathcal{A}_4^{SP_3}}([i,i'][j,j']-SP) &\overset{\text{Lem.5}}{=} C_{\mathcal{A}_4^{SP_3}}(([i,i_1-1],[j,j_1-1])-SP_1) \\ &\quad + C_{\mathcal{A}_4^{SP_3}}(([i_1+1,i'-1][j_1+1,j'-1])-SP_2) \qquad (4.27) \\ &\quad + \sum_{(a'_O,a'_I)\in SP_3\cup\{(a_O,a_I)\}} \text{stemmatch}(a'_O,a'_I) \\ &\overset{\text{Lem.6}}{=} C(([i,i_1-1],[j,j_1-1])-SP_1) \\ &\quad + C(([i_1+1,i'-1][j_1+1,j'-1])-SP_2) \qquad (4.28) \\ &\quad + \sum_{(a'_O,a'_I)\in SP_3\cup\{(a_O,a_I)\}} \text{stemmatch}(a'_O,a'_I) \end{aligned}$$

Note that Lemma 5 is applicable since stem pairs by construction have no open arc matches. Now consider case (4.25e). We consider subsets $\mathcal{A}_5^{a_I}$ of $\mathcal{A}_5$ where for each a_I the set $\mathcal{A}_5^{a_I}$ denotes the subset of alignments A contained in $\mathcal{A}_5$ where a_I is the innermost arc pair of the largest stem pair that is matched by A and whose outermost arc pair ends at (i',j'). Such a stem pair always exists since it must at least contain the arc match $a_O=((i_0,i'),(j_0,j'))$. Furthermore, it is unique since it must be maximal. Consequently the sets $\mathcal{A}_5^{a_I}$ form a partition of $\mathcal{A}_5$ and hence

$$C_{\mathcal{A}_5}(([i,i'][j,j'])-SP) \overset{\text{Lem.4}}{=} \min_{\mathcal{A}_5^{a_I}} C_{\mathcal{A}_5^{a_I}}(([i,i'][j,j'])-SP) \tag{4.29}$$

We fix some a_I and denote $\searrow\!\!a_I$ with (i_1, j_1). Then

$$C_{\mathcal{A}_5^{a_I}}(([i,i'][j,j']) - SP) \overset{\text{Lem.5}}{=} C_{\mathcal{A}_5^{a_I}}(([i,i_1-1][j,j_1-1]) - (SP \cup \{(a_O,a_I)\})) + \text{stemmatch}(a_O,a_I) \tag{4.30}$$

Again Lemma 5 is applicable since (a_O, a_I) is a stem pair and hence has no open arc matches.

Finally consider case (4.25f). Let (a_I, a_O) be the stem pair in SP for which

- $a_O = ((i_1,i'),(j_1,j'))$ if $((i_1,i'),(j_1,j')) \in \text{CR}$
- $a_I = ((i',i_1),(j',j_1))$ if $((i',i_1),(j',j_1)) \in \text{CR}$

There exists at least one such stem pair, since the arc match with end points (i_1, j_1) and (i', j') must be contained in some stem pair of SP because otherwise this arc match would be open in $([i,i'],[j,j']) - SP$. On the other hand there exists at most one such stem pair (a_I, a_O) since due to restriction III of Definition 21 the stem pairs in SP do not overlap. For the uniquely determined (a_I, a_O) the claim $C_{\mathcal{A}_6}(([i,i'],[j,j'])-SP) = C(([i,i_1-1],[j,j_1-1])-(SP-\{(a_O,a_I)\}))$ follows directly from $([i,i'],[j,j']) - SP = ([i,i_1-1],[j,j_1-1]) - (SP - \{(a_O,a_I)\})$ which holds by Definition 21. ∎

Complexity

Let s be the size of a stem pair $((p_{Oa},p_{Ob}),(p_{Ia},p_{Ib}))$, more precisely

$$s := \max\{p_{Ia}^{\text{L}} - p_{Oa}^{\text{L}}, p_{Oa}^{\text{R}} - p_{Ia}^{\text{R}}, p_{Ib}^{\text{L}} - p_{Ob}^{\text{L}}, p_{Ob}^{\text{R}} - p_{Ib}^{\text{R}}\}.$$

We first consider the number of items that are required for the precomputation of the stems. Those items have the form $C(\{[i,p_{I_a}^{\text{L}}],[p_{I_a}^{\text{R}},i']\},\{[j,p_{I_b}^{\text{L}}],[p_{I_b}^{\text{R}},j']\})$ where p_{I_a} and p_{I_b} have n and m possible instances, respectively. For each of them i, i', j and j' can take at most s different values, since they must belong to the same stem pair. Hence, as each item can be computed in constant time, the preprocessing step to align all possible stem pairs can be done in $O(nms^4)$ time and space.

In the main recursion items of the form $C(([i,i'][j,j'])-SP)$ are considered. For i, i', j, and j' as in the previous algorithms we have in total $O(n^2m^2)$ possible instances and with the same trick as before only $O(nm)$ of them need to be maintained in memory at the same time. The number of instances for SP is intuitively related to the number of stem pairs that can be open for a fragment pair at the same time. To make this a little bit more precise we define the notion of crossing number.

Definition 23 (crossing number) For two arc annotated sequence (S_a, P_a) and (S_b, P_b) and some valid partition of $P_a \times P_b$ into CR and NC, the *crossing number* of some point $(x, y) \in [1, |S_a|] \times [1, |S_b|]$ is defined as

$$\text{cross}(x,y) = |\left\{ (a_0,a_I) \in SP_{CR}^{MAX} \,\middle|\, i < x < i' \text{ and } j < y < j' \text{ for } ((i,i')(j,j')) = a_I \right\}|$$

where SP_{CR}^{MAX} denotes the subset of SP_{CR} that contains only pairs of maximal stems (with respect to set inclusion). □

Let k now denote the maximum crossing number among all points of the given two structures. For each open stem pair (a_I, a_O) of a fragment pair $([i, i'], [j, j'])$ either the start points or the end points of both a_I and a_O are contained in the fragment pair. In the first case (a_I, a_O) contributes to the crossing number of the point (i, j) and in the second case it contributes to the crossing number of the point (i', j'). Hence, the total number of maximal open stem pairs for $([i, i'], [j, j'])$ is bounded by $\text{cross}(i, j) + \text{cross}(i', j') \leq 2k$. Each of those maximal stem pairs has $O(s^4)$ smaller stem pairs that it contains and from all of them at most one can be contained in SP simultaneously (due to restriction III of Definition 21). Hence, for SP there exist at most $O((s^4)^{2k}) = O(s^{8k})$ instances for each combination of i, i', j, and j' which results in a space complexity of $O(nms^{8k})$.

Within the recursion each case except (4.25d) and (4.25e) requires only constant time. Case (4.25e) only minimizes over all $O(s^2)$ possible values of a_I since a_O is already determined by (i', j'). Case (4.25d) minimizes over all instances of SP_3 which uniquely determines SP_1 and SP_2 for a fixed SP. Since M_3 has at most $O(s^{8k})$ instances the computation of the in total $O(n^2m^2s^{8k})$ values required $O(n^2m^2s^{8k} \cdot s^8k) = O(n^2m^2s^{16k})$ time.

While at the first sight this complexity does not look like an improvement compared to the algorithm in the previous sections, the size of the parameter k is much smaller for this algorithm than for the previous ones. For simple pseudoknots, for example, and also for many other practical applications (as we will see in the evaluation in Chapter 6) k equals one.

5 Polynomial Alignment of Restricted Classes of Pseudoknots

While the previous chapter presented alignment methods for arbitrary pseudoknots this chapter explores the possibilities to obtain more efficient algorithms for restricted classes of pseudoknots. The presented algorithm scheme that forms the core of this chapter has been published in [38].

5.1 Why Restrictions are both Necessary and Acceptable

The complexity of a DP algorithm depends on the number of items that need to be computed recursively. For RNA structures this is related to the number of fragments that need to be considered. Since each subset of positions of the sequence has a corresponding fragment, the number of possible fragments of a sequence is exponential in its length. Therefore efficient algorithms aim at considering only certain restricted kinds of fragments. An algorithm could for example only consider fragments up to a certain degree (e.g. fragments that contain no or only one gap). Since a fragment of degree k is characterized by $2k$ boundaries, there exist $O(n^{2k})$ fragments with degree at most k for a sequence of length n. Hence a restriction of the degree to some constant k would allow to consider only polynomially many fragments.

The algorithms in the previous chapter have shown that it is important to consider arc-preserving fragments. The difficulty arising from restrictions, for example on the degree, is that one has to guarantee that the pseudoknots can be decomposed into arc preserving fragments that satisfy the respective restriction. For the gap degree restriction this is not possible for arbitrarily complex pseudoknots. To give an intuition for that, the next lemma shows that for arbitrary pseudoknots one cannot guarantee to always find parse trees whose fragments have a degree that is bounded by some constant k.[1]

Lemma 14 (unbounded degree) *For each $k > 0$ there exists an arc annotated sequence (S, P) such that each parse tree for (S, P) contains a fragment with a degree of at least k.* □

[1] Later in this chapter a polynomial alignment algorithm for pseudoknots with parse trees of a bounded degree is presented. Under the assumption that P≠NP this algorithm also implies the claim of this lemma since the alignment task solved by this algorithm is NP-hard for arbitrary pseudoknots. However the constructive proof given here gives a much better intuition.

PROOF Consider the following infinite family of pseudoknot structures $(S^{(i,j)}, P^{(i,j)})$ for $i, j \geq 2$. The element (i, j) of the family consists of a sequence of length $2i \cdot j$ with ij arcs that can be grouped into j different components $P_1^{(i,j)}, \ldots, P_j^{(i,j)}$. Intuitively, each component forms a stem consisting of i arcs, whose endpoints interleave with the two neighboring components. Two examples of the family are visualized in Figure 5.1(a) and (d). Formally, the components $P_k^{(i,j)}$ of $(S^{(i,j)}, P^{(i,j)})$ are defined as

$$P_1^{(i,j)} := \bigcup_{l \in [0,i-1]} \{(l+1, 3i-2l)\}$$

$$P_k^{(i,j)} := \bigcup_{l \in [0,i-1]} \{(1+2l+2i(k-1)-i, 3i-2l+2i(k-1))\} \text{ for } 1 < k < j$$

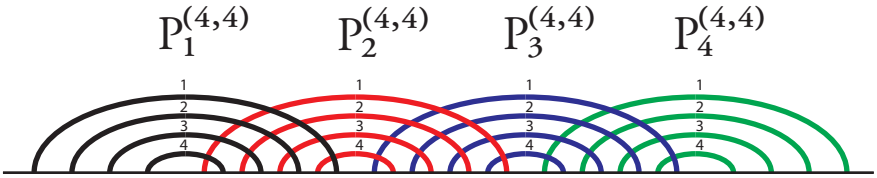
$$P_j^{(i,j)} := \bigcup_{l \in [0,i-1]} \{(1+2l+2i(j-1)-i, 3i-2l+2i(j-1)-(i-l))\}$$

For this family we now show that for $i >> j$, each parse tree of $(S^{(i,j)}, P^{(i,j)})$ contains a fragment with a degree of at least $j-1$.

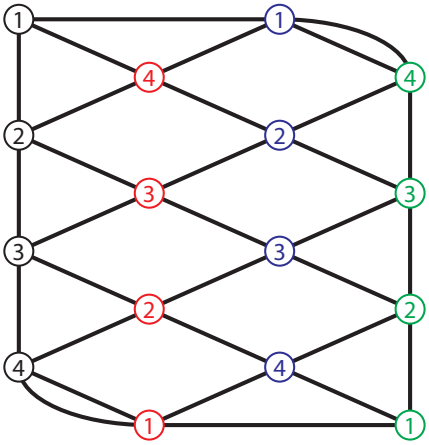
First note, that a fragment of a parse tree of this family can be characterized by the set of arcs $P' \subseteq P^{(i,j)}$ that it contains, since each position is adjacent to an arc. We count the number of boundaries b of such a fragment to determine its degree $b/2$. In order to do this, consider the adjacency graph of $P^{(i,j)}$ which is the graph with node set $P^{(i,j)}$ and an edge between any $p_1, p_2 \in P^{(i,j)}$, if any two ends of the two arcs are directly adjacent in the sequence, i.e. $|x - y| = 1$ for some $x \in \{p_1^{\mathrm{L}}, p_1^{\mathrm{R}}\}$, $y \in \{p_2^{\mathrm{L}}, p_2^{\mathrm{R}}\}$. We add one more arc that connects the arc adjacent to the first position with the arc adjacent to the last position. Two examples for such adjacency graphs are given in Figure 5.1(b) and (c). Due to the correspondence between fragments of a parse tree and subsets of $P^{(i,j)}$, each fragment corresponds to a subset P' of nodes in the adjacency graph. Each edge in the adjacency graph that connects a node in P' with a node that is not contained in P', corresponds to a boundary of the fragment represented by P'. Hence the number of boundaries of the fragment represented by P' corresponds to the size of the cut that separates P' from the remaining graph.

We now first show that each parse tree contains a fragment that contains at least $\frac{1}{4}ij$ and at most $\frac{1}{2}ij$ arcs and then that a cut that separates a fragment of that size from the remaining nodes always has a size of at least $2j-2$. Hence, the corresponding fragment has $2j-2$ boundaries and consequently a degree of $j-1$.

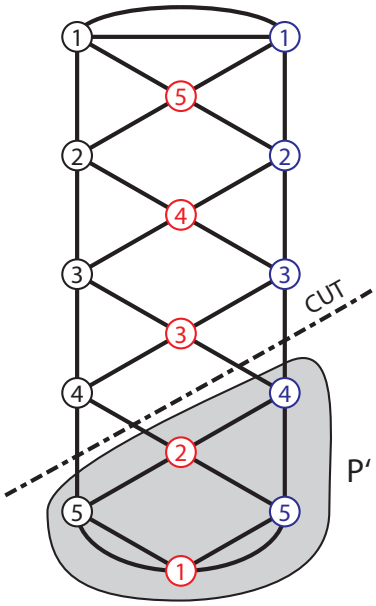
A parse tree always contains a fragment that contains at least $\frac{1}{4}ij$ and at most $\frac{1}{2}ij$ arcs, since the fragment of each inner node in the parse tree contains the union of the arcs of its children and hence in each step the number of contained arcs is at most doubled. More precisely, on each path through the tree from some leaf to the root, the number of arcs contained in the respective fragment at most doubles for each visited node. Since the leaf contains only one arc and the root contains all arcs, there must be some node in between that contains between a quarter and half of all nodes.



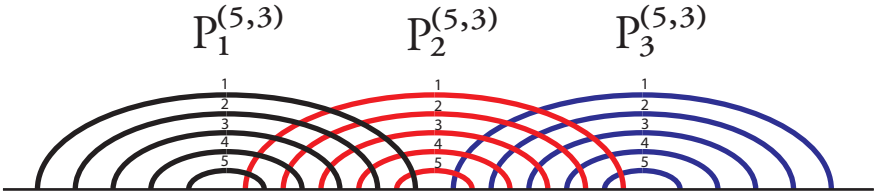
(a) The structure $(S^{(4,4)}, P^{(4,4)})$.



(b) The adjacency graph of the structure $(S^{(4,4)}, P^{(4,4)})$. It indicates which arcs have neighboring end points in the sequence.



(c) The adjacency graph of structure $(S^{(5,3)}, P^{(5,3)})$ with some example set P' and corresponding cut marked.



(d) The structure $(S^{(5,3)}, P^{(5,3)})$

Figure 5.1: Two instances of the family of pseudoknot structures of the proof of Lemma 14 together with their adjacency graphs. The adjacency graphs form a grid whose two dimensions correspond to the two parameters i and j of the family. To obtain the grid like representation, note that the nodes are distributed on the plane such that for each stem the nodes are placed in one column and in each second column the nodes are in reverse order

Consider an instance of the family with $i >> j$. The adjacency graph of the structure forms a two dimensional grid (see Figure 5.1) whose size is i in the one dimension and j in the second. Since $i >> j$, the grid has a line like shape and hence a cut that separates between a quarter and half of all nodes is minimal, if it just cuts one contiguous part from the top or bottom of this line (see Figure 5.1(c)). Such a cut has always a size of at least $2j - 2$. ■

This proof shows on the one hand that one cannot guarantee a bounded degree for arbitrary pseudoknot structures but on the other hand it gives also an intuition that instances with a high degree are rare and difficult to construct. To obtain a high degree, the adjacency graph must form some kind of grid whose size is large in both dimensions, since if the graph has a more linear shape, there exist always small cuts in the graph for sets of arbitrary size. On the other hand, each part of an adjacency graph that forms a two dimensional grid corresponds to a structure according to the motif captured in the family $(S^{(i,j)}, P^{(i,j)})$ in the proof of Lemma 14. In that sense, this motif of a set of stems where neighboring stems maximally interleave is the only "difficult" motif that exists and all other complex structures are just variations of that motif.

Since the restriction on the degree is an important factor to guarantee the polynomial runtime of the algorithm, in this chapter we restrict ourselves to classes of pseudoknots that can be recursively decomposed into fragments with a degree of at most k, where k is some constant. Starting from this central restriction, we will later also identify other restrictions to further reduce the complexity. We will in particular consider several restrictions that various RNA structure prediction algorithms are based on.

From the algorithmic perspective the restrictions make it possible to obtain a polynomial algorithm. But also from the biological perspective there is evidence that such restrictions are reasonable. Rodland [47], for example, states that "allowing arbitrary pseudoknots in secondary structures increases the number of available structures dramatically, but that the vast majority of these are not realistic structures: they are far more heavily knotted than secondary structures found in real life". In particular, stems with interleaved ends as in Figure 5.1a) and c) are not observed in real biological structures and hence restrictions that exclude such structures seem reasonable. Condon et al. [11] have in particular analyzed the restrictions that we are going to focus on and that have been employed by different structure prediction algorithms; they state that "current algorithms do in fact handle a wide range of known biological structures, though not all such structures".

The next section will present a general alignment algorithm scheme that relies on a degree restriction and can be refined with further restrictions. After that we will show how to instantiate this scheme for the various restricted classes of pseudoknots that are considered by different structure prediction algorithms.

5.2 The General Algorithm Scheme

This section describes a general scheme that, depending on how it is instantiated, yields alignment algorithms for different restricted classes of pseudoknots. Similar as in the pseudoknot prediction algorithm scheme in Section 3.2, such a class of pseudoknots is characterized by a set of split types $\mathcal{T}$. A pseudoknot (S, P) is a member of the class $\mathcal{T}$, if there exists a parse tree for (S, P) that contains only splits of types contained in $\mathcal{T}$. Two variants of the algorithm scheme are discussed. First, one that is based on basic split types and second, one that is based on constrained types and requires a more involved recursion.

5.2.1 The Variant for Basic Types

Let $\mathcal{T}$ be a set of basic split types that characterizes the currently considered class of pseudoknots. Then the algorithm takes two arc-annotated sequences (S_a, P_a), (S_b, P_b) and a parse tree for (S_a, P_a) as input[2]. For this parse tree we require that for each inner node F_a and children F_a^1, F_a^2 the split of F_a into F_a^1 and F_a^2 is of some split type $T \in \mathcal{T}$.

The algorithm recursively computes alignments of fragments as follows. Each fragment F_a in the parse tree that is split according to some type T is aligned to all possible fragments F_b of the second sequence that can be split according to the same split type. The computation of this alignment recursively relies on the alignments of the children F_a^1 and F_a^2 to all possible F_b^1, F_b^2, respectively. Hence, the algorithm computes alignments bottom up for all fragments in the parse tree and at the root node arrives at the alignment of the entire structure. Formally, the recursion is captured in the following lemma.

Lemma 15 (split lemma) *Let F_a and F_b be fragments of (S_a, P_a) and (S_b, P_b), respectively. Let (F_a^1, F_a^2) be an arc-preserving split of F_a of basic type T. Then*

$$C(F_a, F_b) = \min_{T\text{-split } (F_b^1, F_b^2) \text{ of } F_b} \left\{ C(F_a^1, F_b^1) + C(F_a^2, F_b^2) \right\} \tag{5.1}$$

□

Proof To capture different cases, as for the proofs in the previous chapter, we first split the set of alignments $\mathcal{A} = \{ A \,|\text{align}_A(F_a, F_b) \}$. We split it into

$$\mathcal{A} = \bigcup_{T\text{-split } (F_b^1, F_b^2) \text{ of } F_b} \mathcal{A}_{(F_b^1, F_b^2)}$$

where $\mathcal{A}_{(F_b^1, F_b^2)}$ is defined as

$$\mathcal{A}_{(F_b^1, F_b^2)} := \left\{ A \in \mathcal{A} \,\middle|\, \text{align}_A(F_a^1, F_b^1) \wedge \text{align}_A(F_a^2, F_b^2) \right\}$$

[2]Such a parse tree can be constructed using standard parsing techniques. Furthermore, also the various structure prediction algorithms discussed in Section 3.2 implicitly construct parse trees that can be reused for this purpose. More details about the parsing are given in the description of the implementation in Chapter 6.

Note that this is no partition, since the different sets $\mathcal{A}_{(F_b^1,F_b^2)}$ may overlap as shown in the example in Figure 3.6. Nevertheless, since the case distinction lemma (Lem 4) does not require the cases to be disjoint, we can apply it as follows:

$$C(F_a, F_b) \overset{\text{Lem.4}}{=} \min_{T\text{-split } (F_b^1,F_b^2) \text{ of } F_b} C_{\mathcal{A}_{(F_b^1,F_b^1)}}(F_a, F_b)$$

The split of F_a into F_a^1 and F_a^2 is arc preserving, since it is contained in a parse tree. Hence, even if the split of F_b into F_b^1 and F_b^2 is not arc preserving, the alignment of the fragment pair (F_a^1, F_b^1) does not contain open arc matches, but at most some open arcs in the second sequence that are not matched to an arc in the first sequence. Hence the two parts are independent and we have

$$\begin{aligned} C_{\mathcal{A}_{(F_b^1,F_b^1)}}(F_a, F_b) &\overset{\text{Lem.5}}{=} C_{\mathcal{A}_{(F_b^1,F_b^1)}}(F_a^1, F_b^1) + C_{\mathcal{A}_{(F_b^1,F_b^1)}}(F_a^2, F_b^2) \\ &\overset{\text{Lem.6}}{=} C(F_a^1, F_b^1) + C(F_a^2, F_b^2) \end{aligned}$$ ∎

The structure of this proof is similar to the proofs for the algorithms shown in the previous chapter and based on the same lemmata. Nevertheless, there is a central difference, namely in the choice of the case distinction. While in the previous chapter each case fixed a certain part of the alignment (for example aligning a certain base to a gap or matching two arcs), this algorithm considers different cases depending on what part of the second sequence is aligned to the two parts F_a^1 and F_a^2 of the first sequence. While the recursions in the previous chapter are entirely symmetric with respect to the two sequences, this algorithm considers a fixed decomposition of the first sequence and the case distinction branches over the different alternatives how to decompose the second sequence accordingly.

Note that the considered splits of the first sequence are always arc preserving since they are contained in the parse tree, while this is not the case for the second sequence. If the split of F_b is not arc-preserving, the respective arcs are broken or removed, since there is no arc of F_a that they can be matched to. The cost for breaking or removing the two ends of the arcs is contained in $C(F_a^1, F_b^1)$ and $C(F_a^2, F_b^2)$, respectively.

The evaluation of the recursion is done by dynamic programming, i.e. all intermediate values $C(F_a, F_b)$ are tabulated, such that each instance is computed only once. The recursive case, shown in Figure 5.2(a), is directly given by Equation (5.1). At the leafs of the parse tree, the base cases shown in Figure 5.2(b) are applied. The actual alignment can be constructed using the usual back-trace techniques.

Complexity Let n and m be the length of the two sequences, respectively. First note that the parse tree has only $O(n)$ nodes, since each split introduces at least one new boundary, of which there exist only $O(n)$ many. The complexity is dominated by the computation at the inner nodes.

The following complexity analysis is similar to the analysis that is done in Section 3.2.2 for pseudoknot prediction algorithms. At each inner node we align some

(a) Recursive case:

$$C(F_a, F_b) = \min_{T\text{-split } (F_b^1, F_b^2) \text{ of } F_b} \left\{ C(F_a^1, F_b^1) + C(F_a^2, F_b^2) \right\},$$

where the parse tree splits F_a into (F_a^1, F_a^2) by a split of basic type T

(b) Base cases:

$$C([i],[l,r]) = \min \begin{cases} C([i],[l+1,r]) + \text{gap}_2(l) & \text{if } l \le r \\ C([i],[l,r-1]) + \text{gap}_2(r) & \text{if } l \le r \\ \text{basematch}(i,l) & \text{if } l = r \\ \text{gap}_1(i) & \text{if } l > r \end{cases}$$

$$C(F_a = ([p^{\mathrm{L}}],[p^{\mathrm{R}}]),\ ([l_1,r_1],[l_2,r_2])) =$$

$$\min \begin{cases} C([p^{\mathrm{L}}],[l_1,r_1]) + C([p^{\mathrm{R}}],[l_2,r_2]) & \\ C(F_a,([l_1+1,r_1],[l_2,r_2])) + \text{gap}_2(l_1) & \text{if } l_1 \le r_1 \\ C(F_a,([l_1,r_1-1],[l_2,r_2])) + \text{gap}_2(r_1) & \text{if } l_1 \le r_1 \\ C(F_a,([l_1,r_1],[l_2+1,r_2])) + \text{gap}_2(l_2) & \text{if } l_2 \le r_2 \\ C(F_a,([l_1,r_1],[l_2,r_2-1])) + \text{gap}_2(r_2) & \text{if } l_2 \le r_2 \\ (\chi(p^{\mathrm{L}},l_1) + \chi(p^{\mathrm{R}},l_2))\frac{w_{am}}{2} & \text{if } (l_1,l_2) = (r_1,r_2) \in P_b \end{cases}$$

Figure 5.2: (a) Recursive case for basic split type and (b) base cases of the algorithm.

fragment F_a with T-split (F_a^1, F_a^2) to all possible F_b that can also be split according to some split (F_b^1, F_b^2) of type T. Since we store the value $C(F_a, F_b)$ for all those F_b, the space complexity depends on the number of instances of F_b which is $\#_P^m(T)$. Since the computation of $C(F_a, F_b)$ minimizes over all possible T-splits (F_b^1, F_b^2) of F_b, the time complexity is given by the number of instances for (F_b^1, F_b^2), i.e. $\#_C^m(T)$.

Since there exist $O(n)$ nodes in the parse tree, the time and space complexity of the algorithm can be bounded as $O(n\#_C^m(T'))$ and $O(n\#_P^m(T''))$, respectively, where T' and T'' are the types contained in the parse tree, for which $\#_C^m(T')$ and $\#_P^m(T'')$ are maximal, respectively.

Given the constants k_p, k_1 and k_2 from Lemma 1 the time complexity of the algorithm is $O(nm^{k_p+k_1+k_2})$ and the space complexity is $O(nm^{2k_p})$. This can be further simplified to $O(nm^{2k})$ and $O(nm^{3k})$, respectively, where k is the maximal degree among all fragments in the parse tree. To get an intuition for this number, note that for most pseudoknots that have been observed in nature there exist parse trees for which $k = 2$. Hence they can be aligned in $O(nm^6)$ time and $O(nm^4)$ space.

5.2.2 An Optimized Variant for Constrained Types

By the preceding complexity analysis, the time and space complexity of the algorithm directly depends on the number of parent instances, $\#_P^m(T)$ and the number of child instances, $\#_C^m(T)$ of the basic types T that occur in the parse tree. Constraints on split types reduce the number of instances, however, the recursion considered in the previous section (Figure 5.2) is not correct for constrained types.

The problem is that if a fragment in the first sequence satisfies some constraints, it is not always the case that the constraints are also satisfied for the fragment of the second sequence which it is aligned to. As an example, consider Figure 5.3 where both intervals of F_a^1 satisfy a length constraint whereas the corresponding intervals of F_b^1 do not. In the example it is visible that the problems occur only if some boundaries in the second sequence are aligned to gaps. More precisely, the intervals in the second sequence might have a length of more than one, but if the corresponding interval in the first sequence consists of only one base, there is at most one match and all other positions are aligned to gaps. Hence, the recursion is extended with additional 'shrink'-cases to 'eat away' the gapped bases. Those 'shrink'-cases can then be applied until the second sequence also satisfies the constraints.

Definition 24 (shrinking) Let (F_b^1, F_b^2) be a T-split of F_b for some constrained type T. A boundary b of F_b is called a *constrained boundary* if it is a boundary of an interval of F_b^1 or F_b^2 that is subject to a length constraint. The set of shrinkings of F_b with respect to T, short $\text{shrinking}_T(F_b)$, is the set of fragments F_b' with the same degree as F_b for which $\hat{F_b'} = \hat{F_b} - \{b\}$ for some constrained boundary b of F_b.□

Consider Figure 5.4 as an example for the set of shrinkings of a split. With this notion, the recursion can be refined for constrained types as follows.

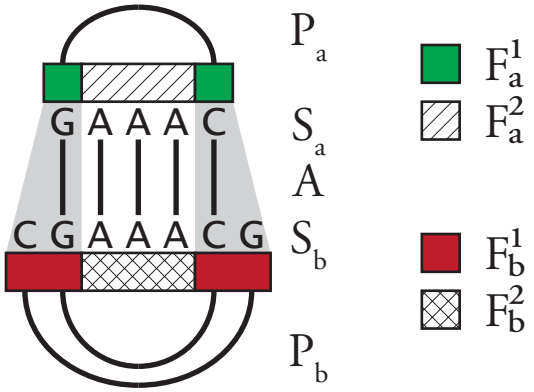


Figure 5.3: The first sequence is split such that both intervals of F_a^1 satisfy a length constraint but there is no appropriate split such that F_b^1 also satisfies those constraints.

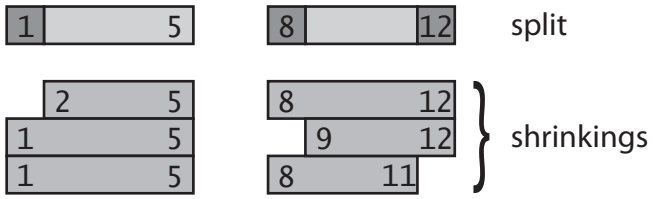


Figure 5.4: The fragment $\{[1,5],[8,12]\}$ has three shrinkings with respect to the constrained split type $1'2G1'21'$, namely $\{[2,5],[8,12]\}$, $\{[1,5],[9,12]\}$ and $\{[1,5],[8,11]\}$.

Lemma 16 (split lemma for constrained types) *Let F_a and F_b be fragments of (S_a, P_a) and (S_b, P_b), respectively. Let (F_a^1, F_a^2) be an arc-preserving T-split of F_a, where T may contain constraints such that*

I no two adjacent intervals both contain length constraints

II if T contains a maximality constraint, the first and last interval have no length constraint.

III T contains only a maximality constraint, if the first and last boundary of F_a and F_b coincide with the beginning and the end of the respective sequence.

Then

$$C(F_a, F_b) = \min \begin{cases} \min\limits_{T\text{-split } (F_b^1, F_b^2) \text{ of } F_b} C(F_a^1, F_b^1) + C(F_a^2, F_b^2) \\ \min\limits_{F_b' \in \text{shrinking}_T(F_b)} C(F_a, F_b') + \text{gap}_b(j) \text{ for the } j \in \hat{F}_b - \hat{F}_b' \end{cases}$$

□

Proof If some constrained boundary of F_b is aligned to a gap then

$$C(F_a, F_b) = \min_{F_b' \in \text{shrinking}_T(F_b)} C(F_a, F_b') + \text{gap}_b(j) \text{ for the } j \in \hat{F}_b - \hat{F}_b'.$$

It remains the case where no constrained boundary of F_b is aligned to a gap. In that case, we construct a T-split (F_b^1, F_b^2) for which

$$C(F_a, F_b) = C(F_a^1, F_b^1) + C(F_a^2, F_b^2)$$

as follows. By Lemma 15 such a split exists and we only need to ensure that there exists also such a split that satisfies all constraints of T. By requirement II and III maximality constraints are always satisfied. If we assume that the split does not satisfy some length constraint, we construct another split that satisfies the length constraint as follows. Since a length constrained interval of the first sequence can be aligned to at most one position of the second sequence, all but at most one position in the corresponding interval in the second sequence are aligned to gaps. Different situations for this are sketched in Figure 5.5. If the interval is not located at a boundary, as shown in Figure 5.5(a) and (b) a new split can be constructed such that all positions aligned to gaps are covered by the neighboring fragments since, by requirement I, those have no length constraints. If, on the other hand, the interval is adjacent to a gap then it is always aligned to the corresponding border in the second sequence, as shown in Figure 5.5c, since otherwise the corresponding boundary of F_b is aligned to a gap, as shown in Figure 5.5d, which is handled by the shrink case. ■

The recursion of Lemma 16 is combined with the same base cases as the previous algorithm (Figure 5.2b) to obtain the final algorithm.

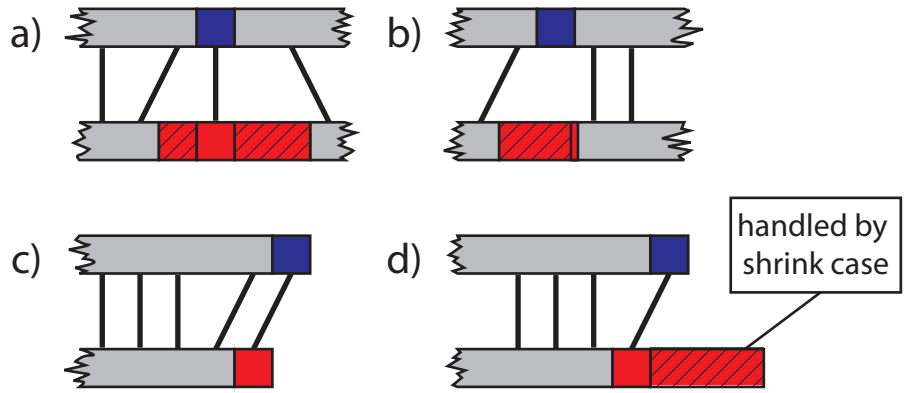


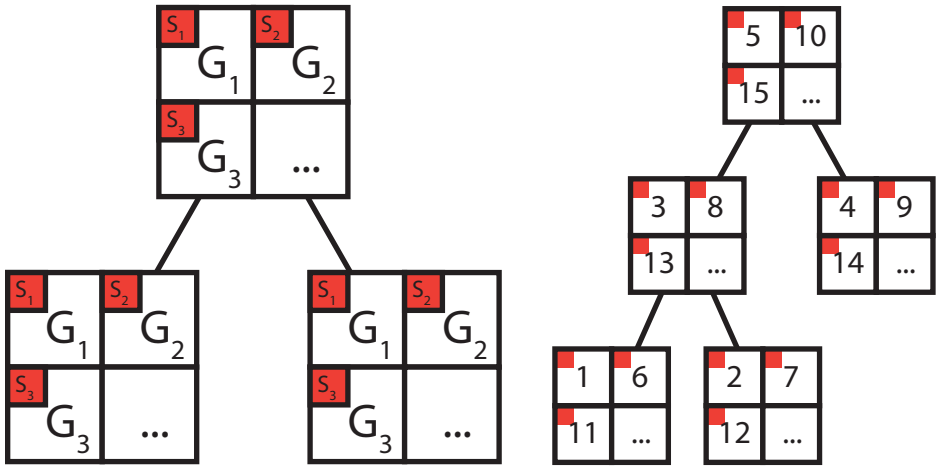
Figure 5.5: Different situations in the proof of Lemma 16. The interval with the length constraint in the first sequence is highlighted in blue and the corresponding interval in the second sequence that does not satisfy the length constraint is shown in red. The black hatching indicates the part of the interval that is removed such that it satisfies the length constraint. In (a) and (b) those parts can be covered by the neighboring intervals, in (d) the shrink case is applied.

Complexity

The complexity analysis is completely analogous to the variant for basic types since the additional shrink case minimizes only over a constant number of values, namely one for each length constraint in T. Consequently the shrink case does not increase the time and space complexity of the algorithm which is hence bounded by $O(n\#_C^m(T'))$ and $O(n\#_P^m(T''))$, respectively, where T' and T'' are again the elements of $\mathcal{T}$ such that $\#_C^m(T')$ and $\#_P^m(T'')$ are maximal. The advantage compared to the variant for basic types is that for constraint types $\#_C^m(T')$ and $\#_P^m(T'')$ are smaller and can be estimated with Lemma 2 instead of Lemma 1. We now show how to improve the space complexity even more for certain instances of the scheme.

Improving Space Complexity with Invariants Analogously to the structure prediction algorithms (see Section 3.2.2), also for the alignment the space complexity can be improved with the help of a grouping according to certain invariants. The grouping idea generalizes from the prediction of one sequence to the alignment of two sequences without difficulty.

So far the recursion was presented in a way that suggested that the computation is done bottom up along the parse tree for one node after the other. A grouping $\mathcal{S}_1 \cdots \mathcal{S}_k \cup \mathcal{G}_1 \cdots \mathcal{G}_k$ partitions the set of parent fragments into simple and grouped fragments such that simple fragments are maintained in memory, whereas the grouped fragments of one group are deleted as soon as the next group is computed



(a) Grouping of a node and its children. (b) Evaluation order of the groups.

Figure 5.6: Grouping of the values that are computed for each node of the parse tree. Part (a) shows the grouping of some node and its two children. The numbers in part (b) show in what order the respective groups are computed.

(see Definition 10 for details). With such a grouping the computations necessary at the different nodes of the parse tree have to be interleaved: First for all nodes of the parse tree (in a bottom up order) only the entries of $\mathcal{S}_1$ and $\mathcal{G}_1$ are computed and then on all nodes the entries of $\mathcal{G}_1$ are deleted before in the next stage $\mathcal{S}_2$ and $\mathcal{G}_2$ are computed for all nodes. This is visualized in Figure 5.6. Note that at each node of the parse tree only one split type is considered and not several ones, as in the recursions for structure prediction. Since for one given split type usually either all instances are grouped or all instances are simple, usually at each node either all sets $\mathcal{G}_i$ or all sets $\mathcal{S}_i$ are empty. Hence a grouping also groups the nodes of the parse tree into simple nodes and grouped nodes. However, we describe the evaluation here such that each node has a (possibly empty) simple part and a (possibly empty) grouped part, to keep it as general as possible and to simplify presentation by avoiding various case distinctions.

Lemma 17 (space complexity for grouped fragments) *Let two arc annotated sequences (S_a, P_a) and (S_b, P_b) be of length n and m, respectively, let t be a parse tree for (S_a, P_a) with a set of split types $\mathcal{T}$ that is invariant with respect to some grouping $\mathcal{S}_1 \cdots \mathcal{S}_k \cup \mathcal{G}_1 \cdots \mathcal{G}_k$. Furthermore, let f be some function that satisfies Properties (3.7) and (3.8) from Section 3.2.*

Then the alignment algorithm scheme can compute the alignment in $O(nf(m))$ space without recomputing each value more than a constant number of times. □

Proof The proof is completely analogous to the proof of Lemma 3 with the only difference that each stage is now done for all nodes of the parse tree which results in

the additional factor n. Since the argumentation why the evaluation order satisfies all relevant properties is at each node exactly the same as for Lemma 3, we just describe the evaluation order precisely and skip the arguments about its properties.

The evaluation order is visualized in Figure 5.6; the evaluation consists of k stages. In each stage i the values for the elements of $\mathcal{S}_i \bigcup \mathcal{G}_i$ are computed for all nodes of the parse tree (bottom up). Before entering stage $i+1$, the space for $\mathcal{G}_i$ is freed on all nodes. In principle each $\mathcal{G}_i$ and also each $\mathcal{S}_i$ can already be deleted for some node as soon as the respective entries for its parent are computed. While this could make a difference in practice, it does not affect the asymptotic space complexity since for arbitrary parse trees nevertheless values of $O(n)$ many nodes have to be kept in memory at the same time. ■

In the next subsection we analyze the complexity of the algorithm for various practical classes of pseudoknots.

5.3 Tailored Instances of the Scheme for all Structure Prediction Algorithms

In this section, we focus on the behavior of the general algorithm scheme for the different classes of pseudoknots predicted by the structure prediction algorithms analyzed in Section 3.2. We show that the alignment can benefit from the structural restrictions in exactly the same way as the prediction does. In particular we show for each of the prediction algorithms how to construct a corresponding alignment algorithm with only a linear increase in complexity (see Table 5.1).

This yields new alignment algorithms for five classes of pseudoknots for which no alignment algorithms existed so far. For the only pseudoknot class for which an alignment algorithm existed before, the new scheme improves the time/space complexity from $O(n^5m^5)/O(n^4m^4)$ to $O(nm^6)/O(nm^4)$. Assuming that the two sequences have an equal length (i.e. $n = m$) this is an improvement by a cubic factor in both time and space. For the simplest class of pseudoknots that is considered, the new algorithm scheme yields a complexity of $O(nm^4)$ time and $O(nm^2)$ space. Also note that on nested structures the algorithm behaves like an algorithm by Jiang *et al.* [27].

R&E structures

The prediction algorithm by Rivas and Eddy [46] requires $O(n^6)$ time and $O(n^4)$ space. As described in Section 3.2.3 the algorithm is based on a set of split types for which $\#_C^m(T') \in O(m^6)$ and $\#_P^m(T'') \in O(m^4)$. For each R&E structure trivially there exists a parse tree that uses only the split types of the R&E algorithm. Hence from the complexity analysis of the alignment algorithm scheme follows that for this class the alignment can be done on $O(nm^6)$ time and $O(nm^4)$ space. Compared to

Table 5.1: Pseudoknot classes and complexity of their prediction and alignment. For A&U the first value is for simple and the second for simple recursive structures.

class		R&E	A&U	L&P	D&P	CCJ	R&G
prediction	time	$O(m^6)$	$O(m^4)/O(m^5)$	$O(m^5)$	$O(m^5)$	$O(m^5)$	$O(m^4)$
	space	$O(m^4)$	$O(m^3)/O(m^3)$	$O(m^3)$	$O(m^4)$	$O(m^4)$	$O(m^2)$
alignment	time	$O(n^5m^5)$	-	-	-	-	-
(literature)	space	$O(n^4m^4)$	-	-	-	-	-
alignment	time	$O(nm^6)$	$O(nm^4)/O(nm^5)$	$O(nm^5)$	$O(nm^5)$	$O(nm^5)$	$O(nm^4)$
(new scheme)	space	$O(nm^4)$	$O(nm^3)/O(nm^3)$	$O(nm^3)$	$O(nm^4)$	$O(nm^4)$	$O(nm^2)$

the algorithm that our scheme yields, the best exact alignment algorithm for this class known so far (by Evans [21]) requires $O(n^5m^5)$ time and $O(n^4m^4)$ space.[3]

A&U structures

The argumentation about the complexity to align A&U structures differs from R&G structures only in the fact that the A&U prediction algorithm (for both simple and simple recursive structures) makes use of an invariant of the considered set of split types to reduce the space requirement according to Lemma 3 (see Section 3.2.3). The same optimization can be done for the alignment according to Lemma 17. Hence, simple A&U structures that are predicted in $O(m^4)$ time and $O(m^3)$ space, are aligned in our scheme in $O(nm^4)$ time and $O(nm^3)$ space. Analogously simple recursive A&U structures for which the prediction requires $O(m^5)$ time and $O(m^3)$ space, can be aligned in $O(nm^5)$ time and $O(nm^3)$ space.

L&P structures

In the prediction algorithm of Lyngsø and Pedersen [31], the only aspect that was non standard with respect to our structure prediction scheme was the grouping used for space optimization. In this grouping the final fragment $[1, n]$ was computed step by step, by minimizing after each stage i only over the instances of the split type 12121 where the second gap of the first child starts at $i-1$ and the gap of the second child ends at i.

For the alignment this computation of the final fragment corresponds to the fact that the root node of the parse tree always has split type 12121. The values computed at the root node can also be computed in such an interleaved fashion: as soon as some stage i for the two children of the root is finished, the root node minimizes over all alignments that consist of fragments computed in this stage. Hence, the space improvement obtained by the grouping can also be used for the alignment which

[3] Evans' algorithm computes the maximum common ordered substructure which can be considered as a special case of the general edit distance measure (see Section 2.2.2).

allows by Lemma 17 to align L&P structures with the alignment scheme in $O(nm^5)$ time and $O(nm^3)$ space.

D&P structures

For D&P structures the argumentation is completely analogous to R&E structures. Without any grouping or other modifications the set of considered split types implies a complexity of $O(nm^5)$ time and $O(nm^4)$ space for the alignment task.

CCJ structures

Also for CCJ structures the time and space complexity of $O(nm^5)$ and $O(nm^4)$, respectively, follows directly from the set of considered split types.

R&G structures

As described in Section 3.2.3 R&G structures are limited to canonical pseudoknots which means pseudoknots that are maximal for the respective sequence. This is a major difference to all previously discussed classes since this criterion is not purely structural but relative to the respective input sequence. A pseudoknot structure may be canonical for one sequence and at the same time be not canonical for another one.

The R&G algorithm computes the optimal structure among all structures that are canonical for the given input. The analogous alignment algorithm computes the optimal alignment among all possible alignments that are canonical for the given input. What is a canonical alignment? For the prediction algorithm the definition of canonical was chosen such that from the $O(n^8)$ instances of the split type 1234153, the split type $1^c23^c41^c53^c$ only considers the $O(n^4)$ instances that are "most reasonable" for the current input. Most reasonable in that case meant that the children 1 and 3 must be maximally extended stems. If we apply the same restriction to the splits of the second sequence that we consider for the alignment, we have only $O(n^2)$ many instances, since in the alignment we have given structures and hence for both the first and third child only $O(n)$ instead of $O(n^2)$ many instances. To resemble the complexity of the prediction algorithm we can consider $O(n^4)$ instead of $O(n^2)$ instances and hence relax the restriction of canonicity. Hence we define the first and third child of $1^c23^c41^c53^c$ to be canonical if and only if their first and last boundary, as well as their second and third boundary are connected by arcs. Then each of the children is determined by those two arcs and hence has $O(n^2)$ instances which results in a total number of $O(n^4)$ instances of $1^c23^c41^c53^c$ and a much weaker restriction of canonicity. Without increasing this number of instances asymptotically, in addition we can also consider the instances where the child 1 or 3 is empty. Then an alignment is canonical, if each stem of a canonical pseudoknot in the first sequence is aligned either to a gap or to a fragment $([l_1, r_1], [l_2, r_2])$ of the second sequence for which $(l_1, r_2) \in P_b$ and $(r_1, l_2) \in P_b$.

One little detail is still missing. The R&G algorithm does not decompose the canonical stems of the split type $1^c23^c41^c53^c$ any further. But to compute the alignment this is necessary since the children 1 and 3 are not atomic. Since those children are parts of canonical stems, they can be further decomposed with the split type $12'G2'1$. As this decomposition relies on no other split types, the first and last boundary both remain invariant. Hence, by a grouping according to those two boundaries the space complexity can be reduced by a quadratic factor. The final time and space complexity for the alignment algorithm of R&G structures is therefore bounded by $O(n^4)$ and $O(n^2)$, respectively.

5.4 Possible Extensions of the Scheme

In this section we briefly sketch possible extensions of the alignment algorithm scheme. The ideas are not described in detail but just point to possible directions of future work.

5.4.1 A Scanning Variant

So far we have assumed to apply the algorithm to two sequences of approximately the same length. But alignment methods can also be used to scan a large structure for a small motif, i.e. to find within the large structure the part that best matches the motif. This can, for example, be realized with a sliding window approach in which a window of the size of the motif is slided over the first sequence and for each position of the window its optimal alignment to the motif is computed according to the algorithm scheme. Finally, among all window positions the one with the best alignment is the one that fits best to the motif.

The problem with such a sliding window approach is that, since the windows overlap, a lot of alignments of fragments are recomputed many times. Therefore, it would be desirable to reuse the results computed for a window position i as much as possible when the window is slid to the next position $i+1$. While this is more an implementation issue than a general problem it nevertheless causes some difficulties. At least a grouping to reduce space complexity will usually conflict with such an optimization for a scanning variant.

5.4.2 A Variant for Combined Alignment and Structure Prediction

The alignment algorithm scheme requires two sequences with given, fixed structures as input. For the first sequence such a structure is necessary to construct the parse tree that determines how the structure is recursively decomposed. However, for the second sequence the presence of the structure is no essential requirement, since the algorithm scheme considers all fragments of it independent from the structure. In the second sequence the structure is only required for the scoring scheme. Hence, the algorithm scheme could be generalized to align one sequence with given structure to another one without given structure.

One possibility would be to assume for the second sequence S_b not a fixed set of base pairs P_b, but a base pair probability matrix that gives for each potential base pair a probability. Those probability matrices for pseudoknot structures could, for example, be computed with the D&P pseudoknot prediction algorithm [17] since this algorithm is not only able to predict the MFE structure but also the partition function. Compared to computing the MFE structure with any prediction algorithm described in Section 3.2 this approach would be much more robust, since it does not commit to the most probable structure but considers all structures simultaneously together with their associated probability.

Only little modifications of the algorithm scheme are necessary in order to work with a base pair probability matrix instead of a fixed structure for the second sequence, since just the scoring scheme has to be adjusted to account for the base pair probabilities. So, whenever two atomic fragments are aligned to each other, there is not either an arc match or an arc breaking or some other case, but only a probability for each of these cases. Hence, the computation at the leaves of the parse tree must be modified such that it combines the probabilities of all those possible cases into one score. For the inner nodes of the parse tree the computation would remain unchanged.

While the approach with base pair probability matrices would be similar to the LocARNA approach [59, 41] also a Sankoff [49] like approach based on a full energy model would be a possible alternative. While the approach with base pair probability matrices is still a two stage approach where first structure probabilities are determined with a prediction algorithm and later on the alignment is computed, the Sankoff algorithm computes the structure and the alignment simultaneously. However, such an extension is less straight forward and would require to compute scores for both the folding and the alignment with exactly the same recursive decomposition.

In principle also the assumption of a given structure for the first sequence could be relaxed. For example, instead of one fixed parse tree the algorithm could also make use of a directed graph that represents several alternative parse trees that are overlaid on top of each other. Of course the complexity of the approach would grow with the number of alternative parse trees but as long as only a small number of highly probable structures is considered, it might still be feasible.

5.4.3 A Partition Function Variant

For RNA structure prediction several approaches do not generate the optimal structure but compute a probability for each possible base pair under the assumption that the ensemble of all structures is Boltzmann distributed [35, 17]. This is more robust than just reporting the optimal structure since in many cases there exist suboptimal structures nearby the optimum that are also relevant in practice.

The same principle can also be applied to the alignment task by computing probabilities for each possible alignment edge. These probabilities can serve then as a local reliability measure for the alignment. Hofacker *et al.* [26] have shown how to

extend the Sankoff algorithm [49] in that way.

Technically, the probabilities in these approaches are computed based on the partition function. The set of all possible alignments (or for structure prediction the set of all possible structures) is assumed to be Boltzmann distributed such that the probability of each alignment is proportional to its Boltzmann weight. While assuming the Boltzmann distribution is reasonable for the ensemble of RNA structures in thermodynamic equilibrium, this assumption is somehow artificial for alignments. In particular the temperature parameter has no physical interpretation but is just an abstract parameter that controls how strong the probability of an alignment is related to its score. Nevertheless the assumption of a Boltzmann distribution has been used for alignments and yields good results in practice [26].

The partition function Z is the sum of the Boltzmann weights of all alignments and can be computed via DP by recursively computing the partition function of fragments. If the recursion to compute the optimal alignment has disjoint cases, the partition function can be computed with the same recursive decomposition. The only difference is that while the computation of the optimum maximizes over different cases and in each case adds the costs of the parts that the recursion descents to, the computation of the partition function adds the Boltzmann weights computed in the different cases and each of those Boltzmann weights is the product of the Boltzmann weights of the parts that the recursion descents to. The requirement that the cases are disjoint is necessary to avoid counting the Boltzmann weights of some alignments more than once. Note that in the current form, the cases of the alignment algorithm scheme are not disjoint: if in some alignment a position of the second sequence is aligned to a gap then splitting directly before this gap and splitting directly behind it are two different cases that both consider this alignment (see Figure 3.6).

The partition function can be used to compute the probability of each single alignment A as $P(A) = \frac{w(A)}{Z}$ where $w(A)$ denotes the Boltzmann weight for A. The probability of a singe alignment edge (i, j) is then

$$\sum_{A \text{ with } (i,j) \in A} P(A) = \frac{\sum_{A \text{ with } (i,j) \in A} w(A)}{Z} \tag{5.2}$$

Similar to the inside outside algorithm for probabilistic context-free grammars [4, 30], the sum of the Boltzmann weights of alignments that contain a certain alignment edge (i, j) can be obtained more efficiently by calculating this weight not for the entire structure directly, but to compute it for some inner fragment (inner probabilities) and a complementary outer fragment (outer probabilities) and multiplying the two values.

In summary, to turn the algorithm scheme into a partition function variant, mainly the following aspects have to be modified: the cases of the recursion have to be made disjoint and an additional recursion has to be developed to compute the outer probabilities (the inner probabilities can be computed analogous to the current recursion of the scheme).

6 Practical Applications

This chapter describes some practical applications of the algorithms developed in this thesis to evaluate their applicability. First, in Section 6.1 implementations for both the alignment methods developed in Chapters 4 and 5 are presented and compared to other existing alignment approaches. Then, in Section 6.2 a pipeline for the detection of pseudoknots in potential non-coding RNAs that is based on the new alignment algorithms is discussed and applied to biological data from two different organismic species.

6.1 Implementations

6.1.1 FPTalign

FPTalign, a prototype of the pseudoknot alignment algorithm presented in Chapter 4, was implemented in C++. The implementation is based on the advanced version of the algorithm including the stem optimization to be as efficient as possible. It uses the general edit distance with the restriction on arc altering as scoring scheme and directly implements the recursion discussed in Section 4.3.3.

To make it applicable to larger instances the program also offers the possibility not to fill the entire matrices of the recursion but to only evaluate a user specified range around the diagonal of the matrices. This is a frequently used heuristic optimization that can be applied to most alignment approaches. If the two aligned sequences have approximately the same length, this heuristic effectively enforces that any position i of one sequence can only be aligned to positions of the other sequence within some range $[i-d, i+d]$ where d is the user specified parameter. If the input sequences have a high similarity, their optimal alignment satisfies this property for rather small d. Hence, often the optimal alignment can be found much faster with such a restriction, however, the optimality of the result is no more guaranteed.

6.1.2 PKalign

The alignment method for restricted classes of pseudoknots described in Chapter 5 was implemented in a program called PKalign. Since this method is able to handle various classes of pseudoknots one major design decision had to be made for the implementation: it could either just focus on one class of pseudoknots or implement the algorithm scheme in its full generality. While the latter option is more flexible and powerful, it is more complex to implement, leaves less room for optimizations and has some overhead due to its generality. Nevertheless, this second alternative

was chosen in order to be able to compare the behavior of the algorithm scheme on the various classes of structures.

As described in Section 5.2, the algorithm requires a parse tree of the first sequence as input. This parse tree has a major impact on the time and space requirement of the algorithm since the essential difference among the various pseudoknot classes is how the parse trees for the respective class look like. The implementation computes a suitable parse tree in a preprocessing step. In practice it is straightforward to find some valid parse tree but it is much harder to find the parse tree on which the algorithm runs fastest. Several parsers that differ in their time and space requirements and in the quality of the resulting parse tree, have been implemented. In other words, some of the parsers take much more time than others but the resulting parse tree might be better in the sense that the subsequent computation of the alignment is much faster. All the parsers where implemented by Jörg Bruder in the context of his diploma thesis [8] that also contains a detailed description and comparison of the various parsers. The evaluation done later in this chapter uses a heuristic parser that turned out to work best on most instances.

As described in Section 5.2.2, the space requirement of the algorithm can be optimized for various pseudoknot classes by reordering the evaluation of the recursion appropriately. As this reordering is different for each structure class, it is difficult to implement in a general fashion. In principle, the parse tree obtained in the preprocessing step would need to be analyzed by the program with respect to the invariants that it satisfies and then an appropriate evaluation order would have to be chosen. Hence, the evaluation order cannot be hard coded in the implementation. To simplify the implementation and to avoid the resulting time overhead, the space optimization was not implemented. This is the biggest drawback of having a universal implementation for all pseudoknot classes instead of an implementation that is tailored to one specific class. In the latter case the respective space optimization could be implemented easily without a time overhead.

Similar to FPTalign, also PKalign allows to limit the computation of the matrices to a range around the diagonal. Again, such a limitation leads to a significant reduction in time and space consumption but, as a heuristic, might yield suboptimal results.

6.1.3 Comparison of both Approaches

Alignment approaches differ in the quality of their results, the class of structures which they can be applied to and in their time and space consumption. Since the two algorithms implemented in PKalign and FPTalign solve the same alignment problem with the same scoring scheme, the quality of their results is always equal and they differ only in the two other aspects.

While FPTalign accepts arbitrary pseudoknots as input, PKalign is restricted to pseudoknots contained in any of the pseudoknot classes described in Section 5.3. Condon *et al.* [11] analyze pseudoknot structures from various databases and report that 434 of the 486 considered structures belong to the R&G class, which is the largest class handled by the algorithm and which contains all the other classes.

Table 6.1: Space requirements and runtime (on a single Opteron 2356 processor with 2.3 GHz) for the alignment of pseudoknot structures from the Rfam database. The column *class* denotes the simplest class of pseudoknots that contains the respective structures, n denotes the average sequence length and k the maximal crossing number, i.e. the exponential factor in the complexity analysis of FPTalign.
⋆For one member of the family, the pseudoknot is not canonical and hence is not contained in the R&G class but only in the A&U class.
⋆⋆For Corona FSE PKalign was only able to align four out of the five instances with the reported 8 GB of memory.

RNA family	class	n	k	time		space	
				FPTalign	PKalign	FPTalign	PKalign
Tymo tRNA	A&U	83	1	0.1 s	2 min 50.6 s	0.5 GB	8 GB
Prion pknot	R&G	41	1	0.8 s	15.0 s	0.5 GB	0.5 GB
Parecho CRE	R&G	112	1	1.0 s	-	0.5 GB	$>$ 8 GB
Antizyme FSE	A&U	58	1	9.2 s	1 min 38.8 s	0.5 GB	4 GB
Corona FSE	R&G⋆	82	1	22.9s	4 min 16.4 s	1 GB	8 GB⋆⋆
Corona pk3	A&U	63	1	29.7 s	1 min 43.4 s	0.5 GB	4 GB
Entero Ori	L&P	122	1	32.8 s	-	1 GB	$>$ 8 GB
Alpha RBS	R&E	110	4	-	-	$>$ 8 GB	$>$ 8 GB

Furthermore, 411 structures belong to the D&P class, which can be aligned by the algorithm scheme one order of magnitude faster. If isolated base pairs that are sometimes considered as tertiary rather than secondary structures are removed, even 483 of the 486 structures belong to the R&G class.

To evaluate time and space consumption of PKalign and FPTalign, several pseudoknot structures from the Rfam database [24] where aligned. A benchmark set of 8 RNA families that are annotated with pseudoknots was used. Albeit in total Rfam contains 16 such families, the test set was restricted to RNAs with a length of at most 125. From each family, five pairs of members where randomly chosen and aligned with both algorithms. The runtime of all five runs was then averaged. To determine the memory consumption, all alignments where computed several times with increasing memory limits, namely 0.5 GB, 1 GB, 4 GB and 8 GB, until a run was successful. For both programs the heuristic option to limit the computation of the matrices around the diagonal was not used.

The results are given in Table 6.1. On all test instances FPTalign performed better than PKalign. The most prominent aspect of the results is that PKalign has a very high memory consumption that makes it impossible to align three of the eight families within the 8 GB limit. In contrast, for FPTalign this is only the case for one instance, Alpha RBS, and most other instances can be aligned even with 0.5 GB of memory. The structure Alpha RBS is a complicated structure in terms of both algorithms: for FPTalign it is the only structure for which the exponential factor k in the complexity estimation has a size larger than one and for PKalign it is the only structure in the test set that falls in the most complex class R&G. While FPTalign

computes all alignments within a few seconds, the running times of PKalign are in the range of a few minutes.

While the results suggest that PKalign is inferior to FPTalign on practical instances this does not imply the same for the underlying algorithms. As described in Section 6.1.2 PKalign is a very general implementation that can handle various restricted classes of pseudoknots. Due to this generality it does not implement the space optimizations that exist for several of those classes. As the evaluation shows the high space consumption is the major drawback of PKalign. Hence, an implementation that is tailored to one specific class of pseudoknots could be much more efficient in practice. Besides the possible space improvement, such an implementation could also be significantly faster since it does not need to implement arbitrary splits and fragments in the general way as PKalign does.

6.1.4 Further Evaluation of FPTalign

Since FPTalign turns out to be the more efficient than PKalign this section describes further tests with FPTalign for larger and more complex pseudoknot structures. The results in this section where published in [37].

For this evaluation alignments of RNA structures of the tmRNA database [64] where computed. We have chosen the longest tmRNA sequence (*Mycobacteriophage Bxz1*, MB), the shortest sequence (*Cyanidium caldarium*, CC), the sequence that contains the largest crossing stems (*Ureaplasma parvum*, UP), and a nested version (UPnest) of the latter, where we removed all left crossing arcs.

We were able to compute the pairwise alignments of these sequences with 1 GB of memory with one exception using 2 GB. Table 6.2 shows that the runtime scales well with the complexity of the involved pseudoknots. As we suggested, the exponential factor k is small on all instances. Whereas alignments of sequences with large pseudoknots take several hours, sequences with small pseudoknots can be aligned in a few minutes. In contrast, sequence length has a much smaller impact on runtime, as in particular the alignments with UPnest show.

As described in Section 4.3.2, the algorithm partitions set of arc pairs into the non-crossing set NC and the crossing set CR and applies to each of the two a different recursion. For the results in Table 6.2 this partition was done according to the left crossing stem criterion (see Lemma 10). However, the runtime can depend heavily on the partition. For example, the alignment of *Ureaplasma parvum* and *Mycobacteriophage Bxz1* took less than three hours if CR was chosen to contain the pairs of left crossing arcs, but more than 6 hours if CR was chosen to contain the right crossing arcs instead. Notably, in this case the better partitioning can be identified in advance by comparing the parameters k and s; k is equal for both cases, s is 10/7 for the left crossing and 12/12 for the right crossing case. This comparison indicates that a more sophisticated partitioning into crossing and nested arc pairs, e.g. greedy or stochastic local optimization, may result in significant speed-ups in practice.

Table 6.2: Runtime of the alignments (on a single Xeon 5160 processor with 3.0 GHz) and the properties of the aligned structures (n=sequence length, s=max. number of arcs in crossing stem, pk=number of pseudoknots, k=fixed parameter of the algorithm) for left crossing partitioning.

aligned sequences	n	s	k	pk	memory	runtime
UP / UP	413/413	10/10	1	4/4	≤ 2 GB	726m 52s
UP / MB	413/437	10/7	1	4/2	≤ 1 GB	172m 53s
UP / CC	413/254	10/2	1	4/1	≤ 1 GB	11m 51s
UP / UPnest	413/413	10/0	0	4/0	≤ 1 GB	4m 43s
MB / MB	437/437	7/7	1	2/2	≤ 1 GB	43m 20s
MB / CC	437/254	7/2	1	2/1	≤ 1 GB	3m 56s
MB / UPnest	437/413	7/0	0	2/0	≤ 1 GB	3m 27s
CC / CC	254/254	2/2	1	1/1	≤ 1 GB	1m 11s
CC / UPnest	254/413	2/0	0	1/0	≤ 1 GB	2m 6s
UPnest/UPnest	413/413	0/0	0	0/0	≤ 1 GB	4m 21s

6.1.5 Comparison to Heuristic Approaches

Besides the algorithms developed in this thesis, for the pairwise alignment of pseudoknot structures there exist only two other algorithms that compute an exact solution of the problem. For these two approaches by Evans [20, 21] no implementations are available. The only implemented approach for pseudoknot alignment it the lara program [5]. In contrast to PKalign and FPTalign, lara is not based on dynamic programming but on integer linear programming (ILP). The integer linear program is solved by lara using Lagrangian relaxation, which is a heuristic approach that does not guarantee an optimal solution. Nevertheless, it gives you a quality guarantee since the method computes not only some alignment but also a bound on how far this alignment is from the optimal one. For some instances this leads to a provably optimal solution. We compared lara to FPTalign to analyze the differences between heuristic and exact approaches.

The evaluation was done as follows. From each of the eight families of the Rfam database used in Section 6.1.3 the two members with the lowest sequence identity where chosen in order to maximally challenge the algorithms. The chosen pairs where aligned with both lara and FPTalign two times: one run with the given pseudoknot structures and one run with only the nested part of the structure. The second run is used to measure the benefit of considering the pseudoknotted structure for the respective algorithm. For FPTalign also two additional runs where computed with different diagonal limitations (as described in Section 6.1.1). The edit costs of FPTalign where set to $w_d = 17$, $w_m = 8$, $w_r = 32$, $w_b = 24$, and $w_{am} = 4$. Since the publicly available version of lara does not accept fixed pseudoknot structures as input, a tailored version of the tool, kindly provided by its authors, was used. The scoring parameters of lara where set as provided by the authors of lara for this test. The quality of the computed alignments was determined by the COMPALIGN score

Table 6.3: Accuracy and runtime of lara and FPTalign. The accuracy is measured with the COMPALIGN score with respect to the hand cured reference alignment of the Rfam database. Values in brackets are for nested input structures. For FPTalign three runs where computed: without diagonal limitation, and with a limitation of 20 and 10.

RNA family	sequence identity	accuracy			
		lara	FPTalign		
			-	20	10
Parecho CRE	83%	1.0000 (1.0000)	0.9868 (0.8728)	0.9868	0.9868
Entero Ori	74%	0.9648 (0.9648)	0.9492 (0.9492)	0.9492	0.1875
Antizyme FSE	70%	0.9739 (0.9217)	0.9304 (0.9304)	0.9304	0.9304
Tymo tRNA	51%	0.9643 (0.9345)	0.9643 (0.9167)	0.9643	0.9643
Prion pknot	51%	0.8537 (0.3902)	0.8537 (0.8537)	0.8537	0.8537
Alpha RBS	48%	0.8605 (0.8605)	- (0.8605)	1.0000	1.0000
Corona FSE	42%	0.6265 (0.5542)	0.6386 (0.5723)	0.6386	0.6386
Corona pk3	42%	0.9600 (0.9600)	0.9120 (0.9120)	0.9120	0.9120

RNA family	time			
	lara	FPTalign		
		-	20	10
Parecho CRE	0.087s (0.080s)	1.137s (0.858s)	0.421s	0.237s
Entero Ori	0.411s (0.383s)	60.128s (0.283s)	4.094s	0.018s
Antizyme FSE	0.037s (0.033s)	9.529s (0.054s)	8.617s	3.482s
Tymo tRNA	0.055s (0.051s)	0.099s (0.037s)	0.074s	0.069s
Prion pknot	0.047s (0.071s)	0.634s (0.016s)	0.619s	0.463s
Alpha RBS	0.079s (0.077s)	- (0.073s)	15m30.186s	4m5.764s
Corona FSE	0.047s (0.051s)	23.114s (0.036s)	20.781s	9.379s
Corona pk3	0.076s (0.087s)	26.469s (0.037s)	26.706s	11.317s

which represents the percentage of columns that are identically aligned as in the reference alignment from the Rfam database. The results are shown in Table 6.3.

The evaluation shows that the quality of the computed alignments is comparable for both approaches. There are instances where FPTalign is slightly better and instances where lara is slightly better. The latter is possible since lara and FPTalign are based on different scoring schemes and hence an optimal solution with respect to one scoring scheme can still be worse than a suboptimal solution with respect to the other scoring scheme. In particular, lara supports affine gap costs which is currently not implemented in FPTalign. A closer inspection of the alignments shows that FPTalign would benefit from affine gap costs in three out of the eight examples (namely *Entero Ori*, *Antizyme FSE*, and *Corona FSE*). Another difference in the scoring is that FPTalign scores all base mismatches with the same constant w_m and all arc mismatches with the same constant w_{am}, whereas lara uses a RIBOSUM matrix [29] that scores a base or arc mismatch depending on the actual bases.

In both approaches the alignment computed for the pseudoknot structures is significantly better than the alignment computed for the nested input structures. A clear example for this is the Prion pknot alignment where lara has a score of 0.8537 for the pseudoknot input but only 0.3902 for the nested input. In the latter case, lara aligns the entire crossing stem wrong by shifting it by one position. Except for Entero Ori, the diagonal limitations for FPTalign do not affect the result.

The runtimes of both approaches are in a range that makes them applicable in practice. Only for the Alpha RBS family which contains a complex pseudoknot consisting of three crossing stems, FPTalign failed in the run without any diagonal limitations due to the memory limit. For nested input structures the runtime for both approaches is comparable and always below one second. For crossing input structures the runtime of FPTalign increases significantly whereas for lara the crossing structures do not increase the runtime. Hence, the presence of pseudoknots makes it harder to find an optimal result, as FPTalign does, but does not affect the runtime of the heuristic approach of lara.

In general the evaluation shows that the exact solution of the alignment problem is feasible with FPTalign but not as fast as the heuristic approach of lara. The results of PKalign are comparable to the results of lara, although the current implementation employs a much simpler scoring scheme without affine gap costs or sophisticated mismatch costs as the RIBOSUM matrix employed by lara. The addition of these features is just an implementation issue but would lead to better results. Further improvements could be obtained by a thorough parameter tuning based on available benchmark sets like the BRAliBase [22], but those are out of the scope of this thesis.

Although lara is faster, the DP method used by FPTalign has advantages in certain scenarios. In contrast to heuristic methods as lara, the DP based methods developed in this thesis do an exhaustive recursive traversal of the search space of possible alignments. This does not only lead to a solution that is guaranteed to be optimal with respect to the scoring scheme, but also allows for extensions like a partition function variant as sketched in Section 5.4.3. Since the partition function requires to sum instead of minimize over the Boltzmann weights of all alignments, this problem cannot be solved by a heuristic search through the search space of potential alignments but requires an exhaustive traversal. Hence, the algorithms developed in this thesis will have most practical relevance due to their potential for such extensions.

Finally, the good behavior of the lara heuristic encourages to investigate additional heuristics to prune the search space of the DP approaches. Besides the already implemented diagonal limitation, there exist also more advances techniques used in other DP algorithms for RNA structures, like for example the sparsification technique that was recently applied to RNA folding [58, 3] and co-folding [62].

6.2 A Pipeline to Detect Conserved Pseudoknots

The reliability of pseudoknot de-novo prediction is still very low. Common prediction programs tend to predict pseudoknots even in pseudoknot free RNA and do not

allow to distinguish safely between true pseudoknots and false positives. Therefore, we developed a prototypical pipeline that combines the de-novo prediction with a subsequent alignment step to identify predicted pseudoknots that are evolutionary conserved among homologous RNAs and contain compensatory base pair mutations. Such a procedure is useful for reliably annotating pseudoknots in unknown RNA, e.g. from genome wide screens for non-coding RNA and yields less false positives than a de-novo prediction on the single sequences. At the current stage the pipeline should be considered as a prototype that shows that pseudoknot alignment can be applied to real biological data in a large scale and that gives a first impression on how frequently conserved pseudoknot structures occur. Not much effort has been put into the tuning of the parameters so far. To improve the quality of the generated results, this is a necessary next step to take.

6.2.1 Detecting Pseudoknots in *Ciona intestinalis*

A first prototype of the pipeline was applied to an RNAz screen of *Ciona intestinalis* [36]. The results of this section where published in [38].

The pipeline starts with a set of homologous RNAs, and performs the following steps: 1.) for each sequence predict locally optimal and suboptimal pseudoknots of the R&G class (using pknotsRG [44] in local mode). 2.) determine candidate pseudoknots that occur at similar positions in k of our sequences (here, $k = 3$). 3.) using PKalign, align the k-tuples of pseudoknots pairwise all-against-all; this information is used to construct a multiple alignment by T-Coffee [40]. 4.) analyze the alignment for conserved, crossing compensatory mutations.

Before applying it to the detection of unknown structures, the approach was tested on the 8 Rfam families that where also used for evaluation in Section 6.1.3. The aim here was to check whether the pipeline is able to reproduce the known pseudoknots. For each family, six sequences where selected randomly for the analysis. Pseudoknot candidates with crossing compensatory mutations where found in all of these families. For four families we could reproduce triplet alignments of the known pseudoknots. Three of these showed crossing compensatory mutations; an example is given in Fig. 6.1a. The figure depicts an alignment of the pseudoknotted sub-sequences with start and end position. For each sub-sequence we show the structure predicted by pknotsRG. The last line gives the consensus structure and highlights base pairs of the pseudoknot which are confirmed by crossing compensatory mutations.

The procedure was then applied to the 50 unannotated ncRNA candidates predicted by an RNAz screen of *Ciona intestinalis* [36]. In this screen, the *C. intestinalis* genome was compared to *C. savignyi* and *O. dioica*, thus per candidate we get three sequences from the three organisms that are analyzed by the above pipeline. Since the RNAz screen looks at both possible reading directions separately, for the pipeline we always considered the reading direction in which the respective candidate was identified. In total, we predicted pseudoknot candidates for only 14 of the 50 RNAs; in contrast, pknotsRG predicts pseudoknots in all of the ncRNAs. Fig. 6.1b shows one prediction by this experiment.

a)

```
                [[-[[[[[....((.....-)).-((((((((((..]]]]]-]].....))))))))))
X90572.1     6 cu-uguacagaaugguaag-cac-guguaguaggagguaca-agcaacccuauugcau 59
                [[-[[[[[....((.....-)).-((((((((((..]]]]]-]].....))))))))))
X66718.1     6 cu-uguacagaaugguaag-cac-guguaaugggagguaca-agcaaccccauugcau 59
                [[[[[[-[.(((...-..)))...((((((((((.-]-]]]]]]-...))))))))))
AF058944.1   6 cucuau-cagauugg-augucuugcugcuauaaua-g-auagag-aagguuauagcag 58

cons. str.      [[-[[[-[.................(((((((((((.-]-]]]-]]....)))))))))))
```

b)

```
                [[[[[[[..((((]]]]]]].(((((((...))))))).......))))--
ci_658349   52 ucucagggugaaaucugagacggaaacgauucguuuccuauauauuuc-- 99
                [[[[[[.(((((((]]]]]]].(((((((...))))))).....)))))))
cs_658349   55 ucucaguuuaauaccugggacggaaacgauucguuuccucuauguauuaa 104
                [[[[[[[.(((((]]]]]]].((-(((.....)))-)).....-)))))-
od_658349   53 ucucagugugacagcugagaccg-uccuacuggga-cgucuau-uguca- 98

cons. str.      [[[[[[[[..(((((]]]]]]].((-(((.....)))-)).....)))))..
```

Figure 6.1: a) Correctly predicted pseudoknot in the Rfam family Corona_pk3 and its alignment. b) Predicted pseudoknot of potential ncRNA.

6.2.2 An Advanced Pipeline for *Drosophilids*

After the tests with the *Ciona intestinalis* data, we applied a refined version of the pipeline to a much larger data set of of *Drosophila melanogaster* and eleven related drosophilid species. The data set consisted of the 163377 high confidence candidates ($p > 0.9$) of the RNAz screen by Rose et al. [48]. Each candidate is based on an alignment of between three and six out of the 12 drosophilid species.

Compared to the pipeline used for the *Ciona intestinalis* data, two changes where made. First, FPTalign was used instead of PKalign, since the evaluation in Section 6.1.3 has shown that it is faster in practice. Second, before aligning the pseudoknots, the vicinity of the pseudoknots was aligned with LocARNA. A new, yet unpublished, partition function variant of this tool was used to determine for each pair of predicted pseudoknots the probability that the fragments containing the pseudoknots are aligned to each other. Only if this probability was above a given threshold, the pseudoknots where aligned with PKalign and extended with the LocARNA alignment to get alignments of the entire candidate sequences. Finally, as in the previous pipeline, the pairwise alignments where combined with T-Coffee to a multiple alignment and analyzed for compensatory mutations.

The RNAz screen whose data was used as input, considered windows of 120 nucleotides length in steps of 40 nucleotides such that neighboring windows overlap 80 nucleotides. Therefore, some RNAz hits occurred in several neighboring windows. In those cases, all windows where taken as input of our pipeline such that for the 16377 RNAz hits in total 23926 windows where considered. Within those 23026 windows, the pipeline could find 906 pseudoknot structures that contained at least one compensatory mutation in each of the crossing stems. From the remaining windows, 18333 contained conserved pseudoknots predictions but they did not show

```
                        [[[[[[[[.((((......]]]]]]]]..))))
                       [[[[[[[[...(((....]]]]]]]]...)))
DroSec_CAF1      1 guccuggugaugccucuaugcaagcauuaccacggauauu 40
                        [[[[[[[[.((((......]]]]]]]]..))))
                       [[[[[[[[...(((....]]]]]]]]...)))
DroSim_CAF1      1 auccugguggugccucuaugcaagcauuaccacggauauu 40
                        [[[[[[[[.((((......]]]]]]]]..))))
                       [[[[[[[[...(((....]]]]]]]]...)))
                       [[[[[[[[........(([]]]]]]]]...))
DroEre_CAF1      1 auccugguaaugccucuaugcaugcguugccacagauauu 40
                        [[[[[[[[.((((......]]]]]]]]..))))
                       [[[[[[[[...(((....]]]]]]]]...)))
3L_DroMel_CAF1   1 auccugguaaugccucuaugcaagcguuaccacggauauu 40
                        [[[[[[[[.((((......]]]]]]]]..))))
                       [[[[[[[[........(([]]]]]]]]...))
                       [[[[[[[[...(((....]]]]]]]]...)))
DroYak_CAF1      1 auccuggcaaugccucuaugcaugcauugccacagauauu 40
cons. str.              [[[[[[[[.((((......]]]]]]]]..))))
```

Figure 6.2: The predicted pseudoknot structures for locus 7542, window 12257.

compensatory mutations in at least one of the two crossing stems. Finally, for 506 windows the predicted pseudoknots could not be aligned within the 4GB memory limit and for 4181 windows the LocARNA alignments showed that none of the predicted pseudoknots where aligned to each other with a probability above the given threshold of 0.001.

The 906 putative pseudoknot structures predicted by the pipeline where then sorted according to the following score. The score is a sum over scores for all compensatory mutated base pairs p that cross at least one other compensatory mutated base pair. Each of these p has a score of $(c+1)^k$, where c denotes the number of compensatory mutated base pairs that cross p and k denotes the number of predicted structures that contain p. A sample prediction is shown in Figure 6.2. As shown in the figure, pknotsRG predicted for each of the sequences several suboptimal pseudoknot structures. While the first stem is similar in all of those predictions and confirmed by four compensatory mutations, only for one prediction of each sequence, the second stem is confirmed by a mutation. Also note that many of the compensatory mutations change some A-U in a wobble base pair G-U or vice versa. This mutation is more probable than others since it is the only mutation that is consistent with the structure and does not require the simultaneous mutation of both bases. Nevertheless, in highly conserved sequences with only few mutations, it is a strong indication for the correctness of the structure. In Figure 6.2, for example, the compensatory mutations are the only present mutations and gaps are not present at all in the considered region.

7 Conclusions

In this thesis we systematically analyzed the problems of RNA pseudoknot structure prediction and alignment. In particular, we focused on the question how dynamic programming algorithms for these problems recursively decompose the RNA structures during their computation.

Compared to pseudoknot free structures the main difference present in all considered algorithms is the use of fragments that contain gaps. Furthermore, the analysis of the existing pseudoknot prediction algorithms showed that there is not just one canonical way for such a decomposition, but a wide range of more or less efficient decomposition strategies that work for certain restricted classes of pseudoknots. We highlighted the similarities of all those approaches by embedding them all in a common scheme that is able to explain the recursive structure and the resulting time and space complexity of the algorithms and abstracts from all the details of the underlying scoring schemes.

For pseudoknot alignment we developed two new approaches. The first approach can handle arbitrary pseudoknots and is fixed parameter tractable. The fixed parameter of this complexity analysis is small on practical instances, namely between one and four for all considered examples. Furthermore, the complexity of the algorithm scales well with the complexity of the pseudoknots given as input. The algorithm can be considered as a generalization of the algorithm of Jiang *et al.* [27] to arbitrary pseudoknot structures.

The second approach is not motivated by previous alignment methods but by the existing DP based structure prediction algorithms. We developed a general scheme that yields for each of the prediction algorithms a corresponding alignment algorithm that covers the same class of structures with only a linear overhead in space and time. For six out of the seven analyzed structure classes no alignment algorithms existed so far. For the last, most complex class, the new scheme yields an alignment algorithm with time and space complexity of (nm^6) and $O(nm^4)$, respectively, whereas the best previous approach has $O(n^5m^5)$ and $O(n^4m^4)$ space complexity (where n and m denote the length of the two sequences, respectively). The fastest among all instances of the scheme has $O(nm^4)$ time and $O(nm^2)$ space complexity and is able to align pseudoknots of the R&G class.

The two new alignment approaches show that, despite the NP-hardness of the general pseudoknot alignment problem, it is feasible for many practical instances and can be solved exactly without the need of an approximation scheme. Both the restriction to certain pseudoknot classes to guarantee a polynomial complexity and the alignment of arbitrary pseudoknots in a fixed parameter tractable framework turned out to be successful. The dissimilarity of the two approaches also shows

that there is a wide range of options of how to compute alignments efficiently. For applications that only consider certain kinds of pseudoknots there is probably a lot of room for specialized algorithms that work well for the instances of interest. The algorithm scheme developed in this thesis does not only cover various such classes of pseudoknots but also forms a flexible starting point for the development of further specialized algorithms as it can probably be extended with additional constraints.

Both alignment methods where implemented to evaluate their applicability in practice. In the current implementation the fixed parameter tractable approach, FPTalign, turned out to be better than the other approach, PKalign, on most instances. However, it is not yet clear whether this difference in performance can be attributed to the underlying algorithms or is just an artifact of the generic implementation of PKalign that covers all considered restricted classes of pseudoknots simultaneously. The quality of the alignments computed by FPTalign and PKalign is comparable to the alignments computed by the only other available pseudoknot alignment tool lara. However, since lara is a heuristic approach without an optimality guarantee for the result, its time and space consumption is less than for the two new approaches.

An important step for future work will be to make the new algorithms more accessible to biologists. For the polynomial alignment method an implementation should be developed that is less generic than PKalign. It should concentrate on just one class of pseudoknots and could therefore be optimized much better to be fast and to require less memory. Furthermore, the implementations should not only be available as command line tools but also via a web front end which is the de facto standard that biologist are used to work with. Another important aspect is the optimization of the parameters of the scoring scheme to yield good results on practical instances.

Finally, several extensions of the polynomial alignment scheme are worthwhile to investigate. Besides a scanning variant and a variant that works with non-fixed input structures, in particular a partition function variant would be of practical relevance.

Bibliography

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

[2] Julien Allali and Marie-France Sagot. A new distance for high level RNA secondary structure comparison. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 2(1):3–14, 2005.

[3] Rolf Backofen, Dekel Tsur, Shay Zakov, and Michal Ziv-Ukelson. Sparse RNA folding: Time and space efficient algorithms. In *Proceedings of the 20th Symposium of Combinatorial Pattern Matching (CPM 2009)*, volume 5577 of *LNCS*, pages 249–262. Springer, 2009.

[4] James K. Baker. Trainable grammars for speech recognition. In *Speech Communication Papers for the 97th Meeting of the Acoustical Society of America*, pages 547–550, 1979.

[5] Markus Bauer, Gunnar W. Klau, and Knut Reinert. Accurate multiple sequence-structure alignment of RNA sequences using combinatorial optimization. *BMC Bioinformatics*, 8:271, 2007.

[6] Tara L. Beattie and Richard A. Collins. Identification of functional domains in the self-cleaving Neurospora VS ribozyme using damage selection. *Journal of Molecular Biology*, 267:830–840, Apr 1997.

[7] Guillaume Blin and Hélène Touzet. How to compare arc-annotated sequences: the alignment hierarchy. In *In 13th Symposium on String Processing and Information Retrieval (SPIRE 2006), volume 4209 of LNCS*, pages 291–303. Springer, 2006.

[8] Joerg Bruder. Design und Evaluierung von Parsing-Techniken für das Sequenz-Struktur-Alignment von RNA mit Pseudoknoten. Diplomarbeit, Albert-Ludwigs-University Freiburg, January 2009.

[9] Ho-Lin Chen, Anne Condon, and Hosna Jabbari. An O(n(5)) Algorithm for MFE Prediction of Kissing Hairpins and 4-Chains in Nucleic Acids. *Journal of Computational Biology*, 16(6):803–15, 2009.

[10] Ramu Chenna, Hideaki Sugawara, Tadashi Koike, Rodrigo Lopez, Toby J. Gibson, Desmond G. Higgins, and Julie D. Thompson. Multiple sequence alignment

with the Clustal series of programs. *Nucleic Acids Research*, 31(13):3497–500, 2003.

[11] Anne Condon, Beth Davy, Baharak Rastegari, Shelly Zhao, and Finbarr Tarrant. Classifying RNA pseudoknotted structures. *Theoretical Computer Science*, 320(1):35–50, 2004.

[12] The ENCODE Project Consortium. Identification and analysis of functional elements in 1% of the human genome by the ENCODE pilot project. *Nature*, 447(7146):799–816, 2007.

[13] Jennifer Couzin. Breakthrough of the year. Small RNAs make big splash. *Science*, 298(5602):2296–7, 2002.

[14] Francis Crick. On protein synthesis. *Symposium of the Society for Experimental Biology*, 12:138–63, 1958.

[15] Francis Crick. Central dogma of molecular biology. *Nature*, 227(5258):561–3, 1970.

[16] Jitender S. Deogun, Ruben Donis, Olga Komina, and Fangrui Ma. Rna secondary structure prediction with simple pseudoknots. In *Proceedings of the second conference on Asia-Pacific bioinformatics (APBC 2004)*, pages 239–246, Darlinghurst, Australia, Australia, 2004. Australian Computer Society, Inc.

[17] Robert M. Dirks and Niles A. Pierce. A partition function algorithm for nucleic acid secondary structure including pseudoknots. *Journal of Computational Chemistry*, 24(13):1664–77, 2003.

[18] Rodney G. Downey and Michael R. Fellows. Fixed-parameter intractability. In *Proceedings of the Seventh Annual Structure in Complexity Theory Conference, 1992.*, pages 36–49, Jun 1992.

[19] Patricia A. Evans. *Algorithms and Complexity for Annotated Sequence Analysis*. PhD thesis, University of Alberta, 1999.

[20] Patricia A. Evans. Finding common subsequences with arcs and pseudoknots. In *Proceedings of the 10th Annual Symposium on Combinatorial Pattern Matching (CPM 1999)*, pages 270–280, London, UK, 1999. Springer-Verlag.

[21] Patricia A. Evans. Finding common rna pseudoknot structures in polynomial time. In *Proceedings of the 17th Annual Symposium on Combinatorial Pattern Matching (CPM 2006)*, volume 4009/2006 of *Lecture Notes in Computer Science*, pages 223–232. Springer Berlin / Heidelberg, 2006.

[22] Paul P. Gardner, Andreas Wilm, and Stefan Washietl. A benchmark of multiple sequence alignment programs upon structural RNAs. *Nucleic Acids Research*, 33(8):2433–9, 2005.

[23] Robert Giegerich. A systematic approach to dynamic programming in bioinformatics. *Bioinformatics*, 16(8):665–77, 2000.

[24] Sam Griffiths-Jones, Alex Bateman, Mhairi Marshall, Ajay Khanna, and Sean R. Eddy. Rfam: an RNA family database. *Nucleic Acids Research*, 31(1):439–41, 2003.

[25] Jakob H. Havgaard, Elfar Torarinsson, and Jan Gorodkin. Fast pairwise structural RNA alignments by pruning of the dynamical programming matrix. *PLoS Computational Biology*, 3(10):1896–908, 2007.

[26] Ivo L. Hofacker and Peter F. Stadler. The partition function variant of sankoff's algorithm. In *Computational Science - ICCS 2004, Part IV*, LNCS 3039, pages 728–735, Heidelberg, June 2004. Springer.

[27] Tao Jiang, Guohui Lin, Bin Ma, and Kaizhong Zhang. A general edit distance between RNA structures. *Journal of Computational Biology*, 9(2):371–88, 2002.

[28] Luca Jovine, Snezana Djordjevic, and Daniela Rhodes. The crystal structure of yeast phenylalanine tRNA at 2.0 A resolution: cleavage by Mg(2+) in 15-year old crystals. *Journal of Molecular Biology*, 301(2):401–14, 2000.

[29] Robert J. Klein and Sean R. Eddy. RSEARCH: finding homologs of single structured RNA sequences. *BMC Bioinformatics*, 4(1):44, 2003.

[30] K. Lari and S. J. Young. The estimation of stochastic context-free grammars using the inside-outside algorithm. *Computer Speech and Language*, 4(1):35–56, 1990.

[31] Rune B. Lyngso and Christian N. S. Pedersen. Pseudoknots in RNA secondary structures. In *Proc. of the Fourth Annual International Conferences on Computational Molecular Biology (RECOMB 2000)*. ACM Press, 2000. BRICS Report Series RS-00-1.

[32] David H. Mathews, Matthew D. Disney, Jessica L. Childs, Susan J. Schroeder, Michael Zuker, and Douglas H. Turner. Incorporating chemical modification constraints into a dynamic programming algorithm for prediction of RNA secondary structure. *Proceedings of the National Academy of Sciences*, 101(19):7287–92, 2004.

[33] John S. Mattick. Challenging the dogma: the hidden layer of non-protein-coding RNAs in complex organisms. *Bioessays*, 25(10):930–9, 2003.

[34] John S. Mattick. The hidden genetic program of complex organisms. *Scientific American*, 291:60–67, Oct 2004.

[35] John S. McCaskill. The equilibrium partition function and base pair binding probabilities for RNA secondary structure. *Biopolymers*, 29(6-7):1105–19, 1990.

[36] Kristin Missal, Dominic Rose, and Peter F. Stadler. Non-coding RNAs in Ciona intestinalis. *Bioinformatics*, 21 Suppl 2:ii77–ii78, 2005.

[37] Mathias Möhl, Sebastian Will, and Rolf Backofen. Fixed parameter tractable alignment of RNA structures including arbitrary pseudoknots. In *Proceedings of the 19th Annual Symposium on Combinatorial Pattern Matching (CPM 2008)*, LNCS, pages 69–81. Springer, 2008.

[38] Mathias Möhl, Sebastian Will, and Rolf Backofen. Lifting prediction to alignment of RNA pseudoknots. In *Proc.of the 13th Annual International Conferences on Computational Molecular Biology (RECOMB 2009)*, volume 5541 of *LNBI*, pages 285–301. Springer, 2009.

[39] Saul B. Needleman and Christian D. Wunsch. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of Molecular Biology*, 48(3):443–53, 1970.

[40] Cedric Notredame, Desmond G. Higgins, and Jaap Heringa. T-Coffee: A novel method for fast and accurate multiple sequence alignment. *Journal of Molecular Biology*, 302(1):205–17, 2000.

[41] Wolfgang Otto, Sebastian Will, and Rolf Backofen. Structure local multiple alignment of RNA. In *Proceedings of German Conference on Bioinformatics (GCB'2008)*, volume P-136 of *Lecture Notes in Informatics (LNI)*, pages 178–188. Gesellschaft für Informatik (GI), 2008.

[42] Cornelis W. Pleij, Krijn Rietveld, and Leendert Bosch. A new principle of RNA folding based on pseudoknotting. *Nucleic Acids Research*, 13(5):1717–31, 1985.

[43] Toolika Rastogi, Tara L. Beattie, Joan E. Olive, and Richard A. Collins. A long-range pseudoknot is required for activity of the Neurospora VS ribozyme. *EMBO Journal*, 15:2820–2825, Jun 1996.

[44] Jens Reeder and Robert Giegerich. Design, implementation and evaluation of a practical pseudoknot folding algorithm based on thermodynamics. *BMC Bioinformatics*, 5:104, 2004.

[45] Jens Reeder, Peter Steffen, and Robert Giegerich. pknotsRG: RNA pseudoknot folding including near-optimal structures and sliding windows. *Nucleic Acids Research*, 35(Web Server issue):W320–4, 2007.

[46] Elena Rivas and Sean R. Eddy. A dynamic programming algorithm for RNA structure prediction including pseudoknots. *Journal of Molecular Biology*, 285(5):2053–68, 1999.

[47] Einar A. Rodland. Pseudoknots in RNA secondary structures: representation, enumeration, and prevalence. *Journal of Computational Biology*, 13(6):1197–213, 2006.

[48] Dominic Rose, Jörg Hackermüller, Stefan Washietl, Kristin Reiche, Jana Hertel, Sven Findeiss, Peter F. Stadler, and Sonja J. Prohaska. Computational RNomics of drosophilids. *BMC Genomics*, 8:406, 2007.

[49] David Sankoff. Simultaneous solution of the RNA folding, alignment and protosequence problems. *SIAM Journal of Applied Mathematics*, 45(5):810–825, 1985.

[50] Bruce A. Shapiro and Kaizhong Z. Zhang. Comparing multiple RNA secondary structures using tree comparisons. *Computational Applications in Biosciences*, 6(4):309–18, 1990.

[51] Sven Siebert and Rolf Backofen. MARNA: multiple alignment and consensus structure prediction of RNAs based on sequence structure comparisons. *Bioinformatics*, 21(16):3352–9, 2005.

[52] Temple F. Smith and Michael S. Waterman. Comparison of biosequences. *Advanced applied Mathematics*, 2:482–489, 1981.

[53] David W. Staple and Samuel E. Butcher. Pseudoknots: RNA structures with diverse functions. *PLoS Biology*, 3(6):e213, 2005.

[54] Gary M. Studnicka, Georgia M. Rahn, Ian W. Cummings, and Winston A. Salser. Computer method for predicting the secondary structure of single-stranded RNA. *Nucleic Acids Research*, 5(9):3365–87, 1978.

[55] Maciej Szymanski, Miroslawa Z. Barciszewska, Volker A. Erdmann, and Jan Barciszewski. A new frontier for molecular medicine: noncoding RNAs. *Biochimica et Biophysica Acta*, 1756(1):65–75, 2005.

[56] Yasuo Uemura, Aki Hasegawa, Satoshi Kobayashi, and Takashi Yokomori. Tree adjoining grammars for RNA structure prediction. *Theoretical Computer Science*, 210:277 – 303, 1999. Paper as Print Copy.

[57] James Watson and Francis Crick. Molecular structure of nucleic acids. a structure for deoxyribose nucleic acid. *Nature*, 171:737–741, 1953.

[58] Ydo Wexler, Chaya Zilberstein, and Michal Ziv-Ukelson. A study of accessible motifs and RNA folding complexity. *Journal of Computational Biology*, 14(6):856–72, 2007.

[59] Sebastian Will, Kristin Reiche, Ivo L. Hofacker, Peter F. Stadler, and Rolf Backofen. Inferring non-coding RNA families and classes by means of genome-scale structure-based clustering. *PLOS Computational Biology*, 3(4):e65, 2007.

[60] Charles Wilson, J. Nix, and Jack Szostak. Functional requirements for specific ligand recognition by a biotin-binding RNA pseudoknot. *Biochemistry*, 37:14410–14419, Oct 1998.

[61] Charles Wilson and Jack W. Szostak. In vitro evolution of a self-alkylating ribozyme. *Nature*, 374(6525):777–82, 1995.

[62] Michal Ziv-Ukelson, Irit Gat-Viks, Ydo Wexler, and Ron Shamir. A faster algorithm for RNA co-folding. In Keith A. Crandall and Jens Lagergren, editors, *Proceedings of the 8th Workshop on Algorithms in Bioinformatics (WABI 2008)*, volume 5251 of *LNCS*, pages 174–185. Springer, 2008.

[63] Michael Zuker and Patrick Stiegler. Optimal computer folding of large RNA sequences using thermodynamics and auxiliary information. *Nucleic Acids Research*, 9(1):133–48, 1981.

[64] Christian Zwieb, Jan Gorodkin, Bjarne Knudsen, Jody Burks, and Jacek Wower. tmRDB (tmRNA database). *Nucleic Acids Research*, 31(1):446–7, 2003.