Optimizing algorithms for the comparative analysis of non-coding RNAs

Dissertation

zur Erlangung des akademischen Grades

doctor rerum naturalium (Dr. rer. nat.)

vorgelegt dem Rat
der Technischen Fakultät
der Albert-Ludwigs-Universität Freiburg

von Diplom-Informatikerin (Bioinformatik)
Christina Otto, geb. Schmiedl

Dekan:
Prof. Dr. Georg Lausen

Prüfungskommission:
Prof. Dr. Andreas Podelski - Vorsitz
Prof. Dr. Fabian Kuhn - Beisitz
Prof. Dr. Rolf Backofen - Gutachter
PD Dr. Björn Voß - Gutachter

Datum der Promotion:
14. Juli 2015

Optimizing algorithms for the comparative analysis of non-coding RNAs

Christina Otto, geb. Schmiedl

2015

Abstract

Non-coding RNAs (ncRNAs) perform essential functions within the cell, such as the regulation of gene expression or catalytic functionalities. Until today, however, the function of most ncRNA molecules is still unknown. As the structure is key to the function of many ncRNAs, much effort has been devoted to the computational structure prediction of ncRNAs and the subsequent functional characterization. This thesis makes important contributions to this field of research by introducing novel fast methods for revealing the functionalities of ncRNA molecules.

The basis of these methods is a novel sparsification technique, the ensemble-based sparsification, which is introduced in the first part of this thesis. Identifying likely structural elements within the structure ensembles of two RNA sequences allows to drastically reduce the search space and leads to a significant shorter runtime. We demonstrate the efficiency of this novel technique for speeding up algorithms for the identification of sequence-structure motifs and simultaneous alignment and folding. However, the applicability of ensemble-based sparsification is not limited to these instances such that this novel technique offers the possibility to speed up other RNA-related tasks in the future as well.

In the second part of this thesis, we introduce the novel method ExpaRNA-P for identifying sequence-structure motifs common to two RNAs in entire Boltzmann-distributed structure ensembles. The core algorithm of the existing approach ExpaRNA solves this problem for a priori known input structures. However, such structures are rarely known; moreover, predicting them computationally beforehand is not an option, since single sequence structure prediction is highly unreliable. In our novel approach ExpaRNA-P, we match and fold RNAs simultaneously, analogous to the well-known simultaneous alignment and folding of RNAs. While this implies much higher flexibility compared with ExpaRNA, the novel approach ExpaRNA-P has the same very low complexity (quadratic in time and space), which is enabled by our novel ensemble-based sparsification. Furthermore, we devise a generalized chaining algorithm to compute compatible subsets of ExpaRNA-P's sequence-structure motifs. We utilize the best chain as anchor constraints for the sequence-structure alignment tool LocARNA, resulting in the very fast RNA alignment program ExpLoc-P. ExpLoc-P is benchmarked in several variants and versus state-of-the-art programs. Across a benchmark set of typical ncRNAs, ExpLoc-P has

similar accuracy to LocARNA but is on average four times faster, while it achieves a speedup over 30-fold for the longest benchmark sequences (≈400nt).

In the third part of this thesis, we present the two novel methods PARSE and SPARSE for simultaneous alignment and folding. PARSE utilizes a lightweight energy model that is derived from a full-featured energy model to score structural contributions. In addition, it integrates Sankoff's original structure prediction flexibility. By utilizing LocARNA's base pair filter, a time complexity of $O(n^4)$ can be obtained for PARSE. Furthermore, we show how the novel ensemble-based sparsification can be applied to derive the sparsified variant SPARSE with a significantly reduced runtime of $O(n^2)$. This means that we introduce the first method with quadratic runtime for simultaneous alignment and folding that does not resort to sequence-based heuristics that could corrupt the alignment quality – as for example the tool RAF does. Furthermore, we demonstrate the effectiveness of our method on benchmarks of real RNA sequences against the state-of-the-art programs LocARNA and RAF. The low computational complexity of SPARSE and RAF is reflected in an overall speedup of around 4 over LocARNA. Whereas RAF's performance drops drastically for instances with low sequence identities, SPARSE benefits from the structure-based optimization and achieves similar alignment quality as LocARNA. Importantly, both tools produce high-quality alignments even for the hard instances with low sequence identity. In addition, we demonstrate the advantage of SPARSE's flexible structure prediction model in comparison with LocARNA. For all sequence identity regions, SPARSE improves LocARNA's structure prediction quality.

In the final part of this thesis, we propose a general theory to describe and implement sparsification in dynamic programming (DP) algorithms. So far, sparsification is mostly a collection of loosely related examples and no general, well understood theory has been developed yet. Our approach is formalized as an extension of algebraic dynamic programming (ADP), which makes it applicable to a variety of algorithms and scoring schemes. In particular, this is the first approach that shows how to sparsify algorithms with scoring schemes that go beyond simple minimization or maximization – as for example the enumeration of suboptimal solutions. On the basis of Nussinov's algorithm, we show how to sparsify RNA structure prediction algorithms.

In summary, this thesis provides novel approaches to decipher the functionalities of ncRNAs. Particularly, we aim at maintaining high quality output while focusing at the same time on making our novel approaches most efficient regarding the runtime. Moreover, we demonstrate in this work the superior performance of our novel methods compared with state-of-the-art programs on real RNA sequences.

Zusammenfassung

Nichtcodierende RNAs (ncRNAs) führen essentielle Aufgaben innerhalb der Zelle aus, wie etwa die Regulierung der Genexpression oder katalytische Funktionen. Jedoch ist bis heute die Funktion der meisten ncRNA Moleküle noch immer unbekannt. Da die Struktur entscheidend für die Funktion vieler ncRNAs ist, wurden große Anstrengungen in die computergestützte Strukturvorhersage von ncRNAs und die anschließende funktionelle Charakterisierung investiert. Diese Arbeit leistet wichtige Beiträge zu diesem Forschungsfeld, indem neue, schnelle Methoden präsentiert werden, die die Funktionsweisen von ncRNA Molekülen aufdecken.

Die Basis dieser Methoden ist eine neuartige Sparsifizierungsmethode, die Ensemble-basierte Sparsifizierung, die im ersten Teil dieser Arbeit eingeführt wird. Die Identifikation wahrscheinlicher struktureller Elemente innerhalb der Strukturensembles von zwei RNA Sequenzen erlaubt es den Suchraum drastisch zu reduzieren und führt zu einer signifikant kürzeren Laufzeit. Wir demonstrieren die Effizienz dieser neuen Methode, indem Algorithmen für die Identifizierung von Sequenz-Struktur-Motiven und die simultane Berechnung von Alignment und Faltung beschleunigt werden. Die Einsetzbarkeit der Ensemble-basierten Sparsifizierung ist jedoch nicht auf diese Anwendungen beschränkt, so dass diese neue Methode die Möglichkeit bietet, in Zukunft auch andere RNA-bezogene Aufgaben zu beschleunigen.

Im zweiten Teil dieser Arbeit präsentieren wir die neue Methode ExpaRNA-P, die Sequenz-Struktur-Motive zwischen zwei RNAs in gesamten Boltzmann-verteilten Strukturensembles identifiziert. Der Kern-Algorithmus des existierenden Ansatzes ExpaRNA löst dieses Problem für a priori bekannte Eingabe-Strukturen. Solche Strukturen sind jedoch selten bekannt; darüber hinaus ist die computergestützte Vorhersage im Voraus keine Lösung, da die Faltung einzelner Sequenzen höchst unzuverlässig ist. In unserem neuen Ansatz ExpaRNA-P wird der Mustervergleich simultan zu der Faltung der RNAs durchgeführt, analog zu der bereits bekannten simultanen Berechnung von Alignment und Faltung von RNAs. Während dies, verglichen mit ExpaRNA, eine viel höhere Flexibilität impliziert, hat der neue Ansatz ExpaRNA-P die gleiche sehr geringe Komplexität (quadratisch in Zeit und Speicherplatz), die durch unsere neue Ensemble-basierte Sparsifizierung ermöglicht wird. Zusätzlich entwickeln wir einen generalisierten Chaining Algorithmus, der kompatible Teilmengen von ExpaRNA-P's Sequenz-Struktur-Motiven berechnet. Wir benutzen die beste Auswahl als Ankerpunkte für

das Sequenz-Struktur-Alignment Tool LocARNA, was zu dem sehr schnellen RNA Alignment Programm ExpLoc-P führt. ExpLoc-P wird in verschiedenen Varianten und im Vergleich zu dem Stand der Technik entsprechenden Ansätzen bewertet. In einem Benchmark typischer ncRNAs erreicht ExpLoc-P eine ähnliche Genauigkeit wie LocARNA, ist aber im Durchschnitt vier mal schneller. Darüber hinaus erzielt es einen über 30-fachen Speedup für die längsten Benchmarksequenzen (≈400nt).

Im dritten Teil der Arbeit stellen wir die zwei neuen Methoden PARSE und SPARSE für die simultane Berechnung von Alignment und Faltung vor. PARSE verwendet ein leichtgewichtiges Energie-Modell welches von einem vollständigen Energie-Modell herrührt, um die strukturellen Beiträge zu bestimmen. Zusätzlich integriert es Sankoff's ursprüngliche Flexibilität der Strukturvorhersage. Indem LocARNA's Basenpaar-Filter verwendet wird, kann eine Zeitkomplexität von $O(n^4)$ für PARSE erreicht werden. Darüber hinaus zeigen wir, wie die neue Ensemble-basierte Sparsifizierung angewendet werden kann, um die sparsifizierte Variante SPARSE mit erheblich reduzierter Laufzeit von $O(n^2)$ zu erzielen. Das bedeutet, dass wir die erste Methode für die simultane Berechnung von Alignment und Faltung mit quadratischer Laufzeit vorstellen, welche nicht auf Sequenz-basierte Heuristiken zurückgreift, die die Alignmentqualität beeinträchtigen können – wie es zum Beispiel das Tool RAF tut. Zusätzlich zeigen wir die Effektivität unserer Methode auf Benchmarks realer RNA Sequenzen im Vergleich zu den Stand der Technik entsprechenden Ansätzen LocARNA und RAF. Die niedrige Komplexität von SPARSE und RAF zeigt sich in einem Gesamt-Speedup von ungefähr 4 im Vergleich zu LocARNA. Während jedoch RAF's Leistungsfähigkeit für Instanzen mit niedriger Sequenzidentität drastisch fällt, profitiert SPARSE von der Struktur-basierten Optimierung und erreicht eine ähnliche Alignmentqualität wie LocARNA. Insbesondere erzeugen beide Tools hoch-qualitative Alignments sogar für die schwierigen Instanzen mit niedriger Sequenzidentität. Wir demonstrieren außerdem den Vorteil des flexiblen Strukturvorhersage-Modells von SPARSE im Vergleich zu LocARNA. SPARSE verbessert die Qualität der Strukturvorhersage von LocARNA für alle Sequenzidentitätsbereiche.

Im letzten Teil dieser Arbeit schlagen wir eine allgemeine Theorie vor, um Sparsifizierung innerhalb von Algorithmen mit dynamischer Programmierung (DP) zu beschreiben und zu implementieren. Bis jetzt ist Sparsifizierung hauptsächlich eine Sammlung von lose zusammenhängenden Beispielen und es wurde noch keine allgemeine, wohlverstandene Theorie entwickelt. Unser Ansatz ist als Erweiterung der algebraischen dynamischen Programmierung (ADP) formalisiert, wodurch er auf eine Vielzahl von Algorithmen und Bewertungsschemata anwendbar ist. Insbesondere ist dies der erste Ansatz, der zeigt wie Algorithmen mit Bewertungsschemata, die über die einfache Minimierung oder Maximierung hinausgehen, sparsifiziert werden können – wie zum Beispiel die Aufzählung von suboptimalen Lösungen. Anhand des Nussinov Algorithmus zeigen wir, wie Algorithmen zur RNA Strukturvorhersage sparsifiziert werden können.

Insgesamt stellt diese Arbeit neue Ansätze bereit, um die Funktionsweisen von ncRNAs zu entschlüsseln. Insbesondere streben wir an, hochwertige Ergebnisse zu erhalten, während wir uns gleichzeitig darauf konzentrieren unsere neuen Ansätze so effizient wie möglich bezüglich der Laufzeit zu gestalten. Darüber hinaus zeigen wir in dieser Arbeit die überlegene Leistungsfähigkeit unserer neuen Methoden verglichen mit dem Stand der Technik entsprechenden Programmen auf echten RNA Sequenzen.

Danksagung

Zuallererst gilt mein Dank Prof. Dr. Rolf Backofen für die Möglichkeit in seiner Gruppe an einem spannenden Thema zu arbeiten und für die Unterstützung, die er mir über die Jahre zuteilwerden lies. Zusätzlich möchte ich mich bei PD Dr. Björn Voß dafür bedanken, dass er als Zweitgutachter meine Arbeit bewertet hat und bei Prof. Dr. Andreas Podelski und Prof. Dr. Fabian Kuhn, dass sie die Prüfungskommission komplettiert haben.

Forschung ist immer eine Gemeinschaftsarbeit und ich konnte mich glücklich schätzen, Inspiration von unterschiedlicher Seite zu bekommen und mit vielen verschiedenen Menschen zusammenarbeiten zu dürfen. Als meine wichtigsten Quellen des Wissens sind Sebastian Will und Mathias Möhl zu nennen. Vielen Dank für die gute Zusammenarbeit und die vielen produktiven Diskussionen. Zu nennen sind außerdem Milad Miladi und Steffen Heyne, die durch ihre Ideen und Impulse maßgeblich zu dieser Arbeit beigetragen haben, sowie Mika Amit und Gad M. Landau, die für den Blick über den Tellerrand gesorgt haben. Sebastian Will, Torsten Houwaart, Milad Miladi, Raphael Otto und Felix Schmiedl möchte ich dafür danken, dass sie Teile meiner Arbeit Korrektur gelesen haben. Mein spezieller Dank gilt außerdem Monika Degen-Hellmuth für die unermüdliche Unterstützung bei allen organisatorischen Fragen.

Bei allen ehemaligen und gegenwärtigen Gruppenmitglieder möchte ich mich für die schöne Zeit bedanken und für das ausgezeichnete Arbeitsklima, das durch die tägliche 4 Uhr Kaffee-Pause und zahlreiche Freizeitaktivitäten abgerundet wurde. Besonders zu nennen sind hier Sita Saunders, Robert Kleinkauf und Daniel Maticzka, die die Zeit wie im Flug vergehen ließen.

Bedanken möchte ich mich auch bei meiner Familie, die es mir ermöglicht hat, meinen Weg zu gehen und mich seelisch und moralisch unterstützt hat, und bei meinen Freunden, die immer für mich da waren und mir über diese stressige Zeit geholfen haben. Zu guter Letzt gilt mein Dank Raphael Otto, dem wichtigsten Menschen in meinem Leben: für deine Unterstützung während des gesamten nicht immer einfachen Doktoranden-Daseins, deine bedingungslose Liebe und deine Geduld, für eine länger als gedachte Zeit in einer Fernbeziehung zu leben.

List of own publications

Basis for this thesis

- Sebastian Will*, **Christina Otto***, Milad Miladi*, Mathias Möhl, and Rolf Backofen. SPARSE: Quadratic time simultaneous alignment and folding of RNAs without sequence-based heuristics. *Bioinformatics*, doi:10.1093/bioinformatics/btv185, first published online April 2, 2015.

- **Christina Otto**, Mathias Möhl, Steffen Heyne, Mika Amit, Gad M. Landau, Rolf Backofen, and Sebastian Will. ExpaRNA-P: simultaneous exact pattern matching and folding of RNAs. *BMC Bioinformatics*, 15(1):404, 2014.

- Sebastian Will*, **Christina Schmiedl***, Milad Miladi, Mathias Möhl, and Rolf Backofen. SPARSE: Quadratic time simultaneous alignment and folding of RNAs without sequence-based heuristics. In *Proceedings of the 17th International Conference on Research in Computational Molecular Biology (RECOMB 2013)*, volume 7821 of *Lecture Notes in Computer Science*, pages 289–290. Springer Berlin Heidelberg, 2013.

- **Christina Schmiedl***, Mathias Möhl*, Steffen Heyne*, Mika Amit, Gad M. Landau, Sebastian Will, and Rolf Backofen. Exact pattern matching for RNA structure ensembles. In *Proceedings of the 16th International Conference on Research in Computational Molecular Biology (RECOMB 2012)*, volume 7262 of *Lecture Notes in Computer Science*, pages 245–260. Springer Berlin Heidelberg, 2012.

- Mathias Möhl*, **Christina Schmiedl***, and Shay Zakov. Sparsification in algebraic dynamic programming. In *Proceedings of the German Conference on Bioinformatics (GCB 2011)*, 2011.

* joint first authors

Further publications

- Mika Amit, Rolf Backofen, Steffen Heyne, Gad M. Landau, Mathias Möhl, **Christina Otto**, and Sebastian Will. Local exact pattern matching for non-fixed RNA structures. *IEEE/ACM Trans. Comput. Biology Bioinform.*, 11(1):219–230, 2014.

- Mika Amit, Rolf Backofen, Steffen Heyne, Gad M. Landau, Mathias Möhl, **Christina Schmiedl**, and Sebastian Will. Local exact pattern matching for non-fixed RNA structures. In *Proceedings of the 23th Annual Symposium on Combinatorial Pattern Matching (CPM 2012)*, volume 7354 of *Lecture Notes in Computer Science*, pages 306–320. Springer Berlin Heidelberg, 2012.

Contents

CHAPTER 1

Introduction

1.1 World of ncRNAs

It was assumed for a long time that RNA molecules – unlike proteins – do not perform active functions within the cell, such as the regulation of gene expression or catalytic activities [CS14]. They were thought to be solely passive carriers of genetic information that aid the translation of DNA into proteins, such as the transfer RNA (tRNA) and ribosomal RNA (rRNA) engaged in the translation of messenger RNA (mRNA) to proteins. All parts of the genome that do not code for proteins – including the intronic regions that are removed by RNA splicing from the pre-mRNA to form the mature mRNA – were regarded "junk" DNA without any specific function [Ohn72]. Initial findings, however, suggest that around 60% of the genome is actually transcribed whereas only about 2% of the genome are covered by protein-coding transcripts [FPM05]. Why should a cell bother to transcribe junk DNA into RNA molecules that do not fulfill any function? A shift in thinking started with the discovery of small nuclear RNAs (snRNAs) that serve as a catalyst in splicing of pre-mRNAs, and small nucleolar RNAs (snoRNAs) that perform rRNA modifications in specific regions that are identified by intermolecular base pairing [MM06, CS14]. Finally, the situation has changed a great deal with the discovery of RNA interference (RNAi) in worms by Fire and Mello [FXM$^+$98] where short double-stranded RNA leads to a silencing of specific genes. RNAi can be induced by microRNAs (miRNAs) that identify the target mRNA via imperfect base pairing and triggers suppression of the translation or its decay [KR08]. Another possible RNAi pathway is mediated by small interfering RNAs (siRNAs) that require perfect complementarity to the target mRNA to induce its cleavage and degradation [KR08]. The possibility to utilize the RNAi mechanism to target and control specific genes soon became apparent and, as a logical consequence, the Nobel prize was awarded to Fire and Mello in 2006. All RNAs that do not code for proteins

are called non-coding RNAs (ncRNAs). A special subset are long ncRNAs (lncRNAs) that are characterized by a length longer than 200 nucleotides. Nowadays, it is understood that the majority of ncRNAs fulfill important cellular tasks [MTF10] and it is assumed that numerous new functions of ncRNAs will be determined in the future [CS14].

Consequently, it is hardly surprising that a lot of diseases are related to ncRNA malfunction [Est11]. Dysregulation of miRNAs, snoRNAs as well as lncRNAs is associated with cancer. Furthermore, disruption of miRNAs is linked to neurological disorders, such as multiple sclerosis, Parkinson's and Alzheimer's disease, and cardiovascular disorders [Est11]. Other types of ncRNAs also play a role in non-cancerous disorders but the analysis is difficult – especially for lncRNAs due to their complexity. Thus, most research has focused on the role of miRNAs in cancer. Therapies that up- or down-regulate those miRNAs that are associated to cancer to restore the original expression patterns have been developed [Est11]. The identification of functional ncRNAs and their exact functional mechanism is the essential precondition for proposing novel therapeutic strategies.

Through the formation of base pairs, which define the secondary structure, an ncRNA molecule can interact with itself, other RNA molecules or proteins. RNA structures guide various cellular processes, including transcription, translation and splicing [WKS+11] and functional characterization was successfully carried out for specific classes of ncRNAs, such as tRNAs, snoRNAs and miRNAs [WKS+11]. Even though computational screens, e.g. [WHS05, PBS+06, SGSM13], identified an abundance of evolutionarily conserved structures in mammalian genomes, functional annotation still lags behind. Since the secondary structure of an ncRNA molecule is key to its function, it is usually more conserved than the primary sequence [TSH+06]. It has been shown for RNA families with sequence conservation below 60% that sequence information alone is not sufficient to uncover evolutionary relationships [GWW05].

With the advent of high-throughput sequencing technologies, pervasive transcription of the human genome was established [CAS+11, The12], which means that "the majority of its bases are associated with at least one primary transcript" [Con07]. Up to 450,000 ncRNAs have been predicted in the human genome [RBT+10]. Additionally, a recent genome-wide analysis in human annotated around 9000 small RNAs and 9500 lncRNA loci [The12]. On top of that, lncRNAs can be kilobases in length [WWH+12].

1.2 General objectives and contributions

The previous section underlined the importance of ncRNAs in regulatory processes. The main insights concerning ncRNAs can be summarized as follows

1. ncRNAs carry out important functions in the cell
2. the functional annotation of ncRNAs lags behind

3. the secondary structure of ncRNAs is key to their function, and
4. a vast amount of ncRNAs that can have a length in the range of kilobases have been found in the human genome.

Based on these findings, we propose novel approaches that can add to the repertoire of methods for the functional annotation of ncRNAs. In bioinformatics, various interesting problems cannot be solved optimally in reasonable time. So algorithm designers have to resort to heuristics to speed up computation. The challenge lies in finding good heuristics that do not significantly compromise the quality and still yield relevant speedup. Note that both the runtime *and* the quality are essential for designing a useful algorithm. How beneficial is an algorithm that provides the result rapidly, but the result cannot be trusted? Or, the other way around, can an algorithm be considered useful that always produces the optimal result but takes ages to finish and thus can only be applied to very small problem instances?

In the light of these considerations, we always consider the combination of runtime and quality. We focus on the efficiency of the proposed algorithms with respect to both the theoretical complexity and the practical speedup that can be obtained on real ncRNA sequences. Crucially, this provides the possibility to process more sequences in a given time frame or deal with longer sequences. Furthermore, we ensure that the applied optimization strategies do not adversely influence the quality, either by proofing the optimality of the results or by showing high accuracy of the results with respect to established quality measures on benchmarks of real RNA sequences.

In particular, we introduce:

- a novel optimization strategy, the ensemble-based sparsification (Chapter 3)
- an algorithm for fast simultaneous exact pattern matching and folding (Chapter 4)
- two algorithms for fast simultaneous alignment and folding (Chapter 5)
- a novel sparsification operator to apply sparsification within the algebraic dynamic programming (ADP) framework [GMS04] (Chapter 6)

1.3 General methods

The theoretical foundations that form the basis of this thesis will be introduced in this section. Dynamic programming is a widely used technique in bioinformatics and constitutes the basis for all novel approaches developed in this thesis. Sparsification was applied to many dynamic programming algorithms in bioinformatics to further speed up the computations. The novel *ensemble-based* sparsification will be introduced in Chapter 3 and subsequently applied in Chapter 4 and 5 to speed up algorithms for the identification of sequence-structure motifs and simultaneous alignment and folding. Chapter 6 describes how the two popular concepts of dynamic programming and sparsification can be combined in a general framework.

1.3.1 Dynamic programming

Dynamic programming (DP) is often used as an optimization strategy to efficiently solve problems with an exponential sized search space. Each DP algorithm is defined by recursions and an objective function that constitutes the scoring scheme and assigns a value to each solution. An optimal solution can for example be one that maximizes a given score or minimizes a given cost. All DP algorithms are based on Bellman's *principle of optimality* that was originally stated as follows:

> *"An optimal policy has the property that, whatever the initial state and initial decision are, the remaining decisions must constitute an optimal policy with regard to the state resulting from the first decision." [Bel57]*

Thus, DP can be applied if the optimal solution can be computed from optimal solutions of subproblems. The optimal solutions of all subproblems are stored and consequently there is no need to recompute them when they are required in a recursion case – thus avoiding combinatorial explosion. For sequence alignment, for instance, there are exponentially many possible alignments for two sequences. The DP algorithm, however, combines optimal solutions for subinstances of the problem to find the optimal alignment in polynomial time. DP algorithms also enjoy great popularity in other areas of bioinformatics, ranging from RNA structure prediction, over predictions of RNA interactions to protein folding.

1.3.2 Sparsification

Although DP allows to compute solutions for many problems in bioinformatics in polynomial time, speed is often still the limiting factor in the analysis of large datasets such that even more efficient approaches are required. Various techniques have been developed to speed up DP algorithms over sequence data. The techniques used comprise the Four-Russians method [GHR80, LMWZU09, FG10], Valiant's approach [Val75, Aku99, ZTZU10], as well as various sparsification approaches [EGGI92, WZZU07, ZUGVWS08, ZUGVWS10, BTZZU11, SMW$^+$10, MSW$^+$10]. Among those techniques, sparsification has been most popular, as it is employable to a wide range of applications, comparably easy to implement, and yields good speedups in both theory and practice. Sparsification approaches consider the intermediate results of an algorithm to identify parts of the search space that can be ignored since they cannot contribute to an optimal solution anymore. As a consequence, the DP tables become sparse and require less computation time and in some cases also less space. Sparsification has been recently applied to RNA structure prediction [WZZU07, BTZZU11], simultaneous alignment and folding [ZUGVWS08, ZUGVWS10], RNA-RNA-interaction prediction [SMW$^+$10], and the prediction of RNA pseudoknot structures [MSW$^+$10]. In all those applications, sparsification yielded a significant speedup. For RNA structure prediction, for example, it reduces the com-

plexity for a sequence of length n from $O(n^3)$ to $O(n^2\psi(n))$, where $\psi(n)$ has been shown to be much smaller than n [WZZU07, BTZZU11].

In contrast, our novel *ensemble-based* sparsification approach, that we will introduce in Chapter 3, discards subsolutions that are unlikely to occur in the respective structure ensembles of the input sequences. We will show that ensemble-based sparsification allows much stronger savings.

1.4 Thesis overview

I contributed fundamentally to all publications that form the basis for this thesis in all phases of the scientific process. In detail, I was involved in defining the theoretical aspects, implementing the required algorithms, setting up evaluation pipelines, generating the results, and writing the manuscripts. This is reflected by the fact that I am a first author for all publications that constitute the basis for this thesis. In general, research requires active discussion and teamwork, and thus can be rarely accomplished by one person alone. Also in this case, other scientists were involved in many stages to make high quality research possible. Thus, "we" is used throughout this thesis to accommodate for that. To be consistent, it is used even for those parts that were conducted solely by myself. In the following, an overview of the topics covered in this thesis is provided.

The current chapter provides the biological background, a first idea about the relevance of the discussed topics and the basic concepts that are used throughout the thesis.

Chapter 2 introduces the necessary technical background and defines the main concepts, where the key points are RNA structure and alignment. Furthermore, an overview over well-established tools for RNA structure and alignment prediction is provided.

In Chapter 3, we introduce a framework for our novel ensemble-based sparsification technique that is based on identifying probabilities that a base or base pair of an RNA is contained in a particular loop. Ensemble-based sparsification constitutes the fundamental element for optimizing algorithms for the identification of sequence-structure motifs (cf. Chapter 4) and simultaneous alignment and folding (cf. Chapter 5).

Chapter 4 introduces the novel algorithm ExpaRNA-P for simultaneous exact pattern matching and folding. This means that motifs are identified in whole Boltzmann-distributed ensembles of two input RNAs. By utilizing the novel ensemble-based sparsification (cf. Chapter 3), we achieve the same time complexity as the less flexible "predecessor" ExpaRNA [HWBB09] that requires fixed structures.

In Chapter 5, we present the novel algorithms PARSE and SPARSE for simultaneous alignment and folding. PARSE combines for the first time the widely used lightweight energy model [HBS04] with the original structure prediction flexibility of Sankoff's algorithm [San85].

The quadratic runtime of the sparsified variant SPARSE results from applying the novel ensemble-based sparsification technique (cf. Chapter 3).

In Chapter 6, we introduce a novel framework that combines the well-established concept of dynamic programming (DP) with the form of sparsification introduced in [WZZU07]. We integrate our novel sparsification operator into the algebraic dynamic programming (ADP) framework [GMS04] – a general framework for defining DP algorithms in a simple and formally precise way. Furthermore, we demonstrate on prominent examples how to easily incorporate sparsification criteria into ADP programs.

CHAPTER 2

Fundamental Concepts

In this chapter, the theoretical foundation is provided that is required for the understanding of the novel approaches discussed in this thesis. A formal definition of RNA structure followed by a description of the main algorithms for RNA structure prediction is given. Furthermore, a definition of sequence and sequence-structure alignment and an overview of existent methods for the computation of sequence-structure alignments is provided.

2.1 RNA structure

A ribonucleic acid (RNA) molecule is a sequence of nucleotides that are linked via phosphodiester bonds. These sugar-phosphate bonds constitute the backbone of the RNA. Nucleotides are built from a ribose sugar linked to a base and a phosphate group. The four bases found in nature are adenine (A), cytosine (C), guanine (G) and uracil (U). Just like in deoxyribonucleic acid (DNA) [WC53], specific bases in the RNA molecule can form hydrogen bonds to create base pairs. The Watson-Crick standard base pairs are G-C and A-U, where G pairs with C and A with U, respectively. Other non-standard base pairs (termed wobble base pairs) can be formed, including the common G-U base pair [Cri66]. Base pairs form between bases of the same RNA and allow manifold conformations of the molecule. RNA structure can be classified in three different categories: the primary structure is the sequence of nucleotides, the secondary structure is defined by the base pairs within the molecule and the tertiary structure considers the actual atom locations in three-dimensional space.

2.1.1 Primary structure

The *primary structure* or *sequence* A of an RNA molecule is a string over the alphabet $\{A,C,G,U\}$. The base at the i-th position of A is denoted by A_i, the substring from posi-

tion i to j by $A_{i..j}$, which is called subsequence in this context. The length of the sequence is denoted by $|A|$. If not stated otherwise, $|A| = n$ holds.

2.1.2 Secondary structure

The secondary structure of an RNA is defined by the set of base pairs, with the constraint that each sequence position can be involved in at most one base pair.

Definition 2.1 (Secondary structure, base pair)
A *secondary structure* R of sequence A is defined by $R \subseteq \{(i,j)|1 \leq i < j \leq |A|\}$ such that for all $(i,j),(i',j') \in R$: $(i = i' \Leftrightarrow j = j')$ and $i \neq j'$. The tuples (i,j) are called *base pairs*.

Some more necessary definitions and important terms related to RNA secondary structure will be given in the following. A position i of A is *paired* with respect to R if it is part of a base pair, i.e. $\exists(i,j) \in R$ or $\exists(j,i) \in R$. Otherwise, position i of A is *unpaired*. The subsequence $A_{i..j}$ is unpaired if k is unpaired for all $k \in [i..j]$ where $[i..j]$ denotes the integer interval from i to j. The *span* of a base pair (i,j) is $j - i + 1$. Position i of a base pair (i,j) is called the left end and position j the right end.

Note that Definition 2.1 does not require that (i,j) is a Watson-Crick or G-U base pair and allows in general rare non-standard base pairs. In most applications, however, the base pairs are restricted to A-U, G-C and G-U. The *nested* or *non-crossing* property in Definition 2.2 can be used to classify RNA secondary structures.

Definition 2.2 (Nested)
A secondary structure R of an RNA sequence A is *nested* or *non-crossing* if there are no base pairs $(i,j),(i',j') \in R$ that are *crossing*, i.e. with $i < i' < j < j'$ or $i' < i < j' < j$.

Secondary structures that do not satisfy the condition in Definition 2.2 are called *crossing*. The shape that is formed by crossing base pairs is called a *pseudoknot*. From a computational point of view, nested RNA secondary structures are much easier to handle as they can be decomposed into five basic secondary structure elements: the hairpin, stacking, bulge, internal and multi-branched loop (see Definition 2.3).

Definition 2.3 (Basic secondary structure elements)
All elements are defined for an RNA sequence A with associated nested secondary structure R.

1. A *hairpin loop* is formed if a base pair (i,j) encloses an unpaired subsequence, i.e. if $A_{i+1..j-1}$ is unpaired and $(i,j) \in R$.
2. A *stacking loop* is formed if two consecutive base pairs occur, i.e. if $(i,j) \in R$ and the interior base pair $(i+1, j-1) \in R$.

3. A *bulge loop* is formed if either the right or left ends of two base pairs are consecutive, i.e. if $(i, j) \in R$ and the interior base pair $(i', j') \in R$ with either $i' = i + 1$ or $j' = j - 1$.
4. An *internal* or *interior loop* is formed if the subsequences between the left ends and right ends of two base pairs are unpaired and have at least length two, i.e. if $(i, j) \in R$ and the interior base pair $(i', j') \in R$ with $i' > i + 1$ and $j' < j - 1$ and $A_{i..i'}$ and $A_{j'..j}$ are unpaired.
5. A *multi-branched loop* or *multiloop* is formed by three or more base pairs. It is closed by base pair (i, j) and has additional interior base pairs $(i_1, j_1) \dots (i_n, j_n)$ with $n > 1$, $i < i_1$, $j_n < j$, $A_{i..i_1}$ and $A_{j_n..j}$ are unpaired, and for all $l \in [1..n-1] : j_l < i_{l+1}$ and $A_{j_l..i_{l+1}}$ is unpaired.

Base pair (i, j) is called the *closing* base pair and the loop is *closed by* (i, j). A *k-loop* is a loop with k base pairs, i.e. the closing base pair plus $k - 1$ interior base pairs. Thus, a hairpin loop is a 1-loop, a stacking, bulge or internal loop is a 2-loop, and a multiloop is a k-loop with $k > 2$. Consecutive stacking loops form a *stem*.

A visualization of the basic RNA secondary structure elements in a nested structure is given in Figure 2.1a. The hairpin loop, the simplest element, is shown in orange. A hairpin loop is a 1-loop as it has one closing and no interior base pair. In this example, base pair $(45, 50)$ closes the hairpin loop. Stacking, bulge and internal loops (shown in light blue, pink and green, respectively) are all 2-loops as they have one closing and one interior base pair. They only differ in the location and number of unpaired bases within the loop: In the stacking, bulge and internal loop, there are – in between the left and right ends of the two base pairs that form the loop – no unpaired bases, unpaired bases only for one of the two, and unpaired bases for both, respectively. In the example provided in Figure 2.1, base pairs $(2, 61)$, $(10, 20)$ and $(27, 38)$ close a stacking, bulge and internal loop, respectively. The corresponding interior base pairs are $(3, 60)$, $(11, 18)$ and $(29, 36)$. The multiloop is the only structural element where the number of interior base pairs is not fixed. The multiloop is thus in general a k-loop with $k > 2$. The multiloop shown in dark blue is a 4-loop as it has the closing base pair $(5, 58)$ and in addition the three interior base pairs $(8, 22)$, $(25, 40)$ and $(42, 53)$.

A simpler representation of a nested RNA secondary structure is the dot-bracket notation.

Definition 2.4 (Dot-bracket notation)
The *dot-bracket notation* is a string of characters '.', '(' and ')' and contains information for each position in the RNA sequence A. If a position in A is unpaired, the position in the dot-bracket notation stores a '.' and if it is the left or right end of a base pair, it stores a '(' and ')', respectively.

All necessary structure information is encoded in the string and it is straightforward to identify for a left end of a base pair the corresponding right end, and vice versa. An example dot-bracket

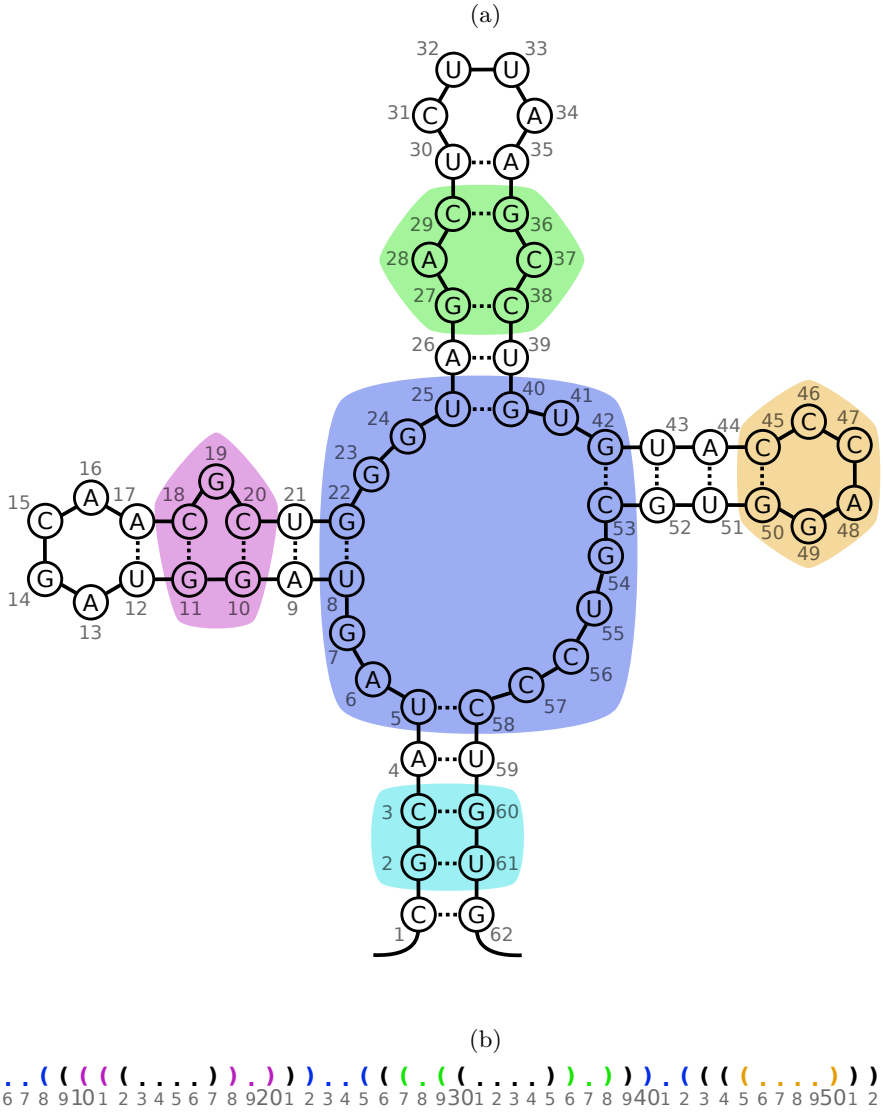


Figure 2.1: **Different visualizations of secondary structure elements.** The hairpin, stacking, bulge, internal and multi loop are highlighted in orange, light blue, pink, green and dark blue. (a) 2D structure representation and (b) dot-bracket notation with annotated structure elements.

notation is given in Figure 2.1b, which is additionally annotated with the secondary structure elements from Figure 2.1a.

A nested RNA secondary structure can be represented by a tree as each base pair has a unique parent.

Definition 2.5 (Parent, loop positions)
The terms are defined for an RNA sequence A with associated nested secondary structure R. A *pseudo base pair* $\psi_A := (0, |A| + 1)$, which covers the whole sequence, is introduced to simplify notation.

a) The *parent of position k in R*, $\text{parent}_R(k)$, is the base pair $(i, j) \in R \cup \psi_A$ with $i < k < j$ such that there is no $(i', j') \in R$ with $i < i' < k < j' < j$. Analogously, the *parent of a base pair* $(i, j) \in R$, $\text{parent}_R(i, j)$, is the parent of i (which is also the parent of j). A position i in A or a base pair $(i, j) \in R$ is *external* iff $\text{parent}_R(i) = \psi_A$.[1] Otherwise it is called *internal*.

b) All unpaired positions and base pairs that have the same parent base pair are denoted by $\text{loop}_R(i, j) = \{k \mid (i, j) = \text{parent}_R(k)\} \cup \{(i', j') \mid (i, j) = \text{parent}_R(i', j')\}$. If $(i, j) \notin R$, $\text{loop}_R(i, j)$ is empty. The expression $\text{loop}_R(i, j)$ combines all bases and base pairs that belong to the same loop closed by (i, j).

In Figure 2.1, position 46 is contained in the hairpin loop (shown in orange) and thus its parent is the closing base pair of the hairpin loop, i.e. base pair $(45, 50)$. All positions and base pairs that are part of the bulge loop (shown in pink) are $\text{loop}_R(10, 20) = \{19, (11, 18)\}$. Note that the closing base pair is not part of the loop it closes. Base pair $(1, 62)$ is the only external base pair in this example. and thus positions 1 and 62 are both external.

Most RNA molecules fold hierarchically, which means that first secondary structure elements are developed that serve later as a basis for more complex interactions, e.g. pseudoknots [TB99]. Mostly, these additional interactions do not significantly alter the already formed structure.

2.1.3 Tertiary structure

The tertiary structure of an RNA is given by the precise location of each atom of the molecule in three-dimensional space. Van der Waals and additional base pair interactions further shape the tertiary structure. Experimental techniques like X-ray crystallography or nuclear magnetic resonance (NMR) spectroscopy can be used to determine exact coordinates of the atoms within the molecule, but they are both time consuming and expensive [SYKB07]. Computational methods that predict the tertiary structure have to cope with the high structural diversity of RNA molecules and are thus usually only applicable to small RNAs or instances that have

[1] Note that we defined the term external in this way to treat the pseudo base pair analogously to every other base pair in the structure. In an alternative definition, a position i can only be external if it is unpaired.

a relatively simple structure [LS10, LS11]. One can say that computational RNA tertiary structure prediction is still at an early stage, but a first step towards more efficient and accurate approaches would be a deep understanding of RNA secondary structure.

2.2 Algorithms for RNA structure prediction

Throughout this thesis, only nested secondary structures are considered and called structures from now on. Efficient algorithms for RNA structure prediction have been studied extensively in the past 40 years. The basic Nussinov algorithm predicts the structure for a given sequence by maximizing the number of base pairs [NPGK78]. However, the likelihood that a structure is formed is mainly controlled by the *Gibbs free energy*, called free energy from now on. It is computed by $E = H - \mathcal{T}S$, where H is the enthalpy, $\mathcal{T}$ the temperature and S the entropy. In this context, this energy is computed as the difference ΔE of the structure compared with the unfolded sequence. The Zuker algorithm [ZS81] computes the structure with the minimum free energy (mfe), which is usually more accurate than maximizing the number of base pairs. However, computing a single mfe structure might not be the best approach as RNA molecules can have multiple possible conformations with almost equal energy. Thus, going beyond Zuker's algorithm, McCaskill's algorithm allows to compute the probability that a base pair is present in the structure of a given sequence by considering a set of structures that are weighted according to their free energy [McC90]. Note that the valid base pairs are typically restricted to G-C, A-U and G-U for all structure prediction algorithms.

2.2.1 Nussinov's algorithm

The Nussinov algorithm [NPGK78] maximizes for an RNA sequence A the number of non-crossing base pairs. The algorithm applies the principle of dynamic programming and recursively computes the maximum number of base pairs for all subsequences from position i to j for $i < j$:

$$\hat{N}(i,j) = \max \begin{cases} \hat{N}(i,j-1) \\ \max\limits_{\substack{h \text{ with } i \leq h < j \text{ and} \\ (h,j) \text{ valid base pair}}} \hat{N}(i,h-1) + \hat{N}(h+1,j-1) + 1 \end{cases} \tag{2.1}$$

The dynamic programming matrix $\hat{N}$ is initialized with $\hat{N}(i,i-1) = 0$, $\hat{N}(i,i) = 0$ for all i. In the first recursion case, position j is unpaired and $\hat{N}(i,j-1)$ is considered without adding a base pair score. The second case maximizes over all possibilities to introduce a base pair with fixed right end j and variable position h as a left end. Due to the nested condition, this splits the problem into two subproblems, namely the part before – $\hat{N}(i,h-1)$ – and below the inserted base pair – $\hat{N}(h+1,j-1)$. Furthermore, a base pair score of 1 is added for the inserted

base pair. The subsequences are processed in a way that all necessary smaller subsequences are already computed when evaluating a subsequence. Since there are $O(n)$ possible split points h, Nussinov's algorithm has $O(n^3)$ time and $O(n^2)$ space complexity. After filling the whole dynamic programming matrix, the optimal structure can be derived by doing a traceback from matrix cell $(1, n)$ with Equation 2.1.

2.2.2 Zuker's algorithm

The structure with the maximal number of base pairs found by Nussinov's algorithm is not necessarily a stable structure as no thermodynamic data is included in the computation. Adding more base pairs to the structure does not necessarily increase the stability of the structure. The Zuker algorithm [ZS81] computes the mfe structure of an RNA sequence by incorporating experimentally derived free energy contributions for the different loop types [MSZT99, TM10], see also Definition 2.3. A stacking loop for example has a negative free energy and thus stabilizes the structure, whereas a large unpaired loop region has a positive free energy and destabilizes the structure. The original version of Zuker's algorithm was introduced without individual energy contributions for multiloops [ZS81]. A variant was introduced in [ZS84], where multiloop energies are expressed by $E_M = a + (k-1)b + uc$ for a k-multiloop with u unpaired bases and a, b and c are constants. In this energy model, the energy for a multiloop is only dependent on the number of interior base pairs and unpaired bases. Using this simplified model allows to construct efficient dynamic programming recursions. Note that the recursions in Equation 2.2 are rephrased in comparison with the original formulation in [ZS84] to make the recursions more similar to the McCaskill recursions introduced later and to have a non-ambiguous representation. This means that there is a one-to-one correspondence between the traceback path within the matrices and the derived structure.

$$W(i,j) = \min \begin{cases} W(i,j-1) \\ \min\limits_{i \le h < j} W(i,h-1) + W^b(h,j) \end{cases}$$

$$W^b(i,j) = \min \begin{cases} E_H(i,j) \\ \min\limits_{i < i' < j' < j} E_{SBI}(i,j,i',j') + W^b(i',j') \\ \min\limits_{i < h < j} W^m(i+1,h-1) + W^{m1}(h,j-1) + a \end{cases}$$

$$W^m(i,j) = \min_{i \le h < j} \begin{cases} c(h-i) + W^{m1}(h,j) \\ W^m(i,h-1) + W^{m1}(h,j) \end{cases}$$

$$W^{m1}(i,j) = \min_{i<h\leq j} W^b(i,h) + b + c(j-h)$$

$$\begin{aligned} E_H(i,j) &: \text{energy of a hairpin loop (experimentally derived)} \\ E_{SBI}(i,j,i',j') &: \text{energy of a stacking, bulge or interior loop} \\ &\quad \text{(experimentally derived)} \\ a+(k-1)b+uc &: \text{energy of a } k\text{-multiloop with } u \text{ unpaired bases} \end{aligned} \tag{2.2}$$

The matrices are initialized for all i and for $j = i$ and $j = i-1$ with: $W(i,j) = 0, W^b(i,j) = W^m(i,j) = W^{m1}(i,j) = \infty$. Matrix W stores the minimum free energy for all subsequences $A_{i..j}$ without any restrictions on the structure and is similar to matrix $\hat{N}$ in Nussinov's algorithm where the maximum number of base pairs is stored. As the energy is minimized, the case that h and j form a base pair needs to be further partitioned in order to assign an energy value according to loop type. For this reason, matrix W^b is introduced, which stores the minimum free energy for all subsequences $A_{i..j}$ with the constraint that i and j are paired to each other. In W^b, case one considers a 1-loop (hairpin loop), case two a 2-loop (stacking, bulge or internal loop) and case three a multiloop ($k > 2$). For scoring the multiloop correctly, matrix W^m is introduced, which has the additional requirement that at least one base pair is predicted. To avoid ambiguity, the split point h in W^m is chosen to always cut off the right-most base pair. For keeping the complexity low, an auxiliary matrix W^{m1} is introduced that stores the minimum free energy under the constraint that i is paired to some position $h \leq j$ and $A_{h..j}$ is unpaired. In W^m, the part before the split point h can be either unpaired (first case) or there can be an additional base pair (case two). In both matrices W^m and W^{m1}, the subsequences are scored according to the multiloop energy. The constant c is added in matrices W^m and W^{m1} each time an unpaired base is introduced. The constant b is added in matrix W^{m1} as a base pair is inserted. The energy contribution a for closing the multiloop is considered in W^b. Usually, the loop length of 2-loops is restricted to a fixed size (typically 30 bases [LBHZS+11]) to achieve the same complexity bounds as the simpler Nussinov algorithm, i.e. $O(n^3)$ time and $O(n^2)$ space complexity.

2.2.3 McCaskill's algorithm

One main achievement of McCaskill's algorithm [McC90] is the computation of both structure and base pair probabilities for a given sequence. In contrast to the simpler Zuker algorithm that computes a single mfe structure, base pair probabilities provide information about all possible conformations of an RNA sequence. Each possible structure R of an RNA sequence A is assigned a probability proportional to its *free energy or Boltzmann factor* $e^{-\mu E(R)}$, where $\mu = \frac{1}{k_B \mathcal{T}}$, $\mathcal{T}$ is the temperature and k_B is a constant. Then, the partition function Z is defined

as the sum of all Boltzmann factors of all possible structures R of sequence A:

$$Z = \sum_{R} e^{-\mu E(R)} \tag{2.3}$$

All structures considered by the partition function form the *Boltzmann ensemble* of sequence A and the structures within the ensemble are *Boltzmann-distributed*. The recursions in Equation 2.2 can be reused with some modifications: the Boltzmann factor replaces the free energy function and each summation is converted into a multiplication and each minimization into a summation. This yields the recursions shown in Equation 2.4. The non-ambiguity of the recursion cases is crucial for the accuracy of the result, as otherwise the same structure would be considered multiple times and the result would be distorted. This non-ambiguity is ensured by considering the split point h in Q^m only if it cuts off the right-most base pair. In comparison, an ambiguous formulation of Zuker's algorithm would still yield the correct result as the single structure with minimum free energy is selected.

$$Q(i,j) = \sum \begin{cases} Q(i,j-1) \\ \sum\limits_{i \le h < j} Q(i,h-1) \cdot Q^b(h,j) \end{cases}$$

$$Q^b(i,j) = \sum \begin{cases} e^{-\mu E_H(i,j)} \\ \sum\limits_{i < i' < j' < j} e^{-\mu E_{SBI}(i,j,i',j')} \cdot Q^b(i',j') \\ \sum\limits_{i < h < j} Q^m(i+1,h-1) \cdot Q^{m1}(h,j-1) \cdot e^{-\mu a} \end{cases}$$

$$Q^m(i,j) = \sum_{i \le h < j} \begin{cases} e^{-\mu c(h-i)} \cdot Q^{m1}(h,j) \\ Q^m(i,h-1) \cdot Q^{m1}(h,j) \end{cases}$$

$$Q^{m1}(i,j) = \sum_{i < h \le j} Q^b(i,h) \cdot e^{-\mu b} \cdot e^{-\mu c(j-h)}$$

$$\begin{aligned} E_H(i,j) &: \text{energy of a hairpin loop (experimentally derived)} \\ E_{SBI}(i,j,i',j') &: \text{energy of a stacking, bulge or interior loop} \\ &\quad \text{(experimentally derived)} \\ a + (k-1)b + uc &: \text{energy of a } k\text{-multiloop with } u \text{ unpaired bases} \end{aligned} \tag{2.4}$$

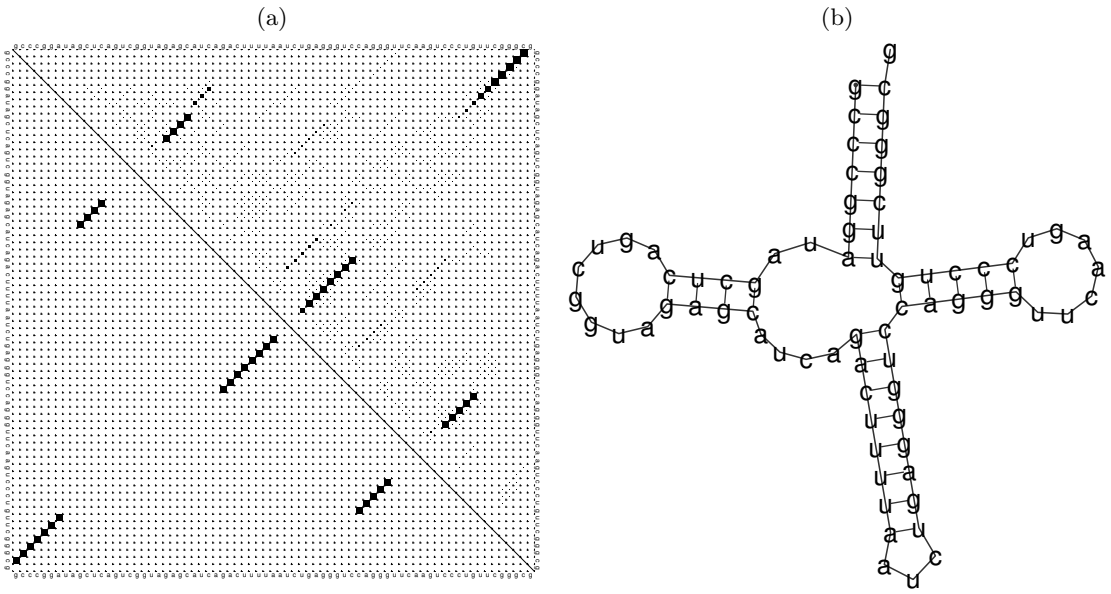


Figure 2.2: **Dot plot and secondary structure representation.** Both representations are generated by RNAfold [LBHZS+11]. (a) Visualization of the base pair probabilities in a dot plot. The base pair probabilities are displayed in the upper right half and the mfe structure in the lower left half. The area of the squares is proportional to the base pair probability. All sides of the rectangle are annotated with the RNA sequence. (b) Corresponding drawing of the mfe structure.

The matrices are initialized for all i and for $j = i$ and $j = i - 1$ with: $Q(i,j) = 1, Q^b(i,j) = Q^m(i,j) = Q^{m1}(i,j) = 0$. Matrix Q stores the partition function for all subsequences $A_{i..j}$. Matrix Q^b contains the partition function for all $A_{i..j}$ with the constraint that i and j form a base pair. Analogously to the Zuker algorithm, matrices Q^m and Q^{m1} are used to efficiently cover the multiloop case with proper scoring (cf. Section 2.2.2). When restricting the length of 2-loops to a fixed size, the partition function can be calculated in $O(n^3)$ time and $O(n^2)$ space. In an alternative formulation, the energy calculation of 2-loops – more precisely interior loops – can be simplified to consider only the number of unpaired bases within the loop and not the position of the interior base pair [McC90]. This constraint also allows to reduce the runtime to $O(n^3)$ and is less restrictive than having a maximal loop length size. By utilizing the partition function $Z = Q(1,n)$ of the full sequence, the probability of a structure R of sequence A can be calculated by $\Pr[R|A] = \frac{e^{-\mu E(R)}}{Z}$. More importantly, the probability of a base pair (i,j) within the structure ensemble can be computed by summing up the probabilities of all structures that include the base pair:

$$\Pr[(i,j)|A] = \sum_{\substack{R \text{ with} \\ (i,j) \in R}} \Pr[R|A]$$

Thus, $\Pr[(i,j)|A]$ is the probability that a random structure R, drawn from the Boltzmann ensemble of A, contains the base pair (i,j). The set of all possible base pairs for RNA sequence A is denoted by P^A. The base pair probabilities can be efficiently computed from McCaskill's dynamic programming matrices in $O(n^3)$ time and $O(n^2)$ space starting from long base pairs and recursing to shorter ones [McC90]. The base pair probabilities are much more expressive than the mfe structure as they contain information of the whole structure ensemble. A widely used tool for predicting RNA structures is RNAfold [BBB+08] that is part of the ViennaRNA Package [HFS+94, LBHZS+11]. It computes the mfe structure based on Zuker's algorithm and the base pair probabilities based on McCaskill's algorithm. The base pair probabilities can be visualized in a *dot plot* where alternative structures can be easily detected and compared against each other. An example dot plot is given in Figure 2.2a with the mfe structure shown in the lower left half. A different visualization of the mfe structure is given in Figure 2.2b. The cloverleaf structure is characteristic of tRNA sequences. Suboptimal structures can be inferred from the base pair probabilities shown in the upper right half in Figure 2.2a. The area of the squares is proportional to the base pair probability. Anti-diagonals in the dot plot correspond to stems in the structure. In a different suboptimal configuration, another stem can be added to the left stem by introducing a bulge loop while shortening the upper stem.

2.3 Sequence and sequence-structure alignment

Similarities between two sequences can be efficiently identified through the computation of an *alignment* where evolutionary relationships can be revealed. The simplest form is the *sequence alignment* where only sequence information is utilized. In a *sequence-structure alignment*, structural features are additionally taken into account.

Throughout this thesis, in addition to sequence A of length n with structure R, we use sequence B of length $|B| = m$. For discussing the computational complexity of the presented algorithms, we assume w.l.o.g. $m \leq n$. A structure of B is denoted by T.

2.3.1 Sequence alignment

Homologous sequences that are derived from a common ancestor are usually pretty similar and a high alignment score is obtained. During evolution, a mutation of a nucleotide leads to a substitution of a single character in an RNA sequence. Furthermore, insertions of nucleotides into the sequence or deletions of nucleotides from the sequence might take place. Generally, the more time has elapsed since their common origin, the more dissimilar the sequences are. This is due to a larger number of edit operations (mutations, insertions and deletions) that can occur during the longer time period. To measure the relatedness of two sequences, the cheapest way to transform the first sequence into the second one is calculated. For that purpose, specific

costs are associated with each edit operation. A formal description of sequence alignment is given in Definition 2.6.

Definition 2.6 (Sequence alignment)
A *sequence alignment* $\mathcal{A}$ *of sequences* A *and* B contains edges between one character of each sequence, i.e. $\mathcal{A} = \{(i,k)|i \in [1..n], j \in [1..m]\}$. Alignment edges do not *cross*, i.e. for all $(i,k),(i',k') \in \mathcal{A} : (i < i' \implies k < k'$ and $i = i' \iff k = k')$. A position i in sequence A is *deleted by* $\mathcal{A}$ iff $\nexists k$ with $(i,k) \in \mathcal{A}$. Analogously, a position k in sequence B is *inserted by* $\mathcal{A}$ iff $\nexists i$ with $(i,k) \in \mathcal{A}$. Positions that are neither deleted nor inserted by $\mathcal{A}$ are *covered by* $\mathcal{A}$. Two positions i and k are *matched to each other by* $\mathcal{A}$ and form a *base match* iff $(i,k) \in \mathcal{A}$. A gap in $\mathcal{A}$ is a maximal stretch of adjacent deleted or inserted positions.

DP provides an efficient way to determine the optimum over the exponentially many alignments between two sequences, as the alignment of the complete sequences can be constructed from optimal alignments of subsequences. The first description of an algorithm for sequence alignment with cubic runtime was given in [NW70] that allowed arbitrary gap costs. Variants with quadratic runtime were introduced in [San72] where no gap costs are considered and in [Sel74] with more general gap costs. In principle, one can either compute the alignment with maximal similarity or minimal distance. Here, only the first variant is shown for the simple linear gap cost scheme. In this case, an alignment is scored by $\text{score}(\mathcal{A}) = \sum_{(i,k)\in\mathcal{A}} \sigma(i,k) + n_{\text{indel}}^{\mathcal{A}}\gamma$, where $\sigma(i,k)$ denotes the similarity between A_i and B_k, $n_{\text{indel}}^{\mathcal{A}}$ the number of gap positions in the alignment and γ ($\gamma \leq 0$) is the fixed cost for each position in the gap. The recursion to compute the sequence alignment with maximal score is given in Equation 2.5. Matrix $K(i,k)$ stores the maximal similarity of subsequences $A_{1..i}$ and $B_{1..k}$. Three cases need to be considered for computing $K(i,k)$: i and k are matched and thus $(i,k) \in \mathcal{A}$, k is inserted (not matched to a position in A) or i is deleted (not matched to a position in B). With this model, the sequence alignment can be computed in $O(n^2)$ time and space [Sel74].

$$K(i,k) = \max \begin{cases} K(i-1,k-1) + \sigma(i,k) & \text{(match)} \\ K(i,k-1) + \gamma & \text{(insertion)} \\ K(i-1,k) + \gamma & \text{(deletion)} \end{cases} \tag{2.5}$$

Favoring longer stretches of insertions and deletions rather than single distributed insertions and deletions is biologically more reasonable. To model this, affine gap costs are introduced where a fixed *gap opening cost* β is added in addition to the fixed gap cost γ for each position in the gap, in this context called *gap extension cost*. A sequence alignment algorithm with affine gap costs was introduced in [Got82].

This procedure can be extended to compute a *multiple sequence alignment* for more than two sequences – a problem that is known to be NP-hard [Jus01]. The progressive alignment strategy

is the most widely used heuristic to efficiently construct a multiple sequence alignment [EB06]. A distance matrix is computed by pairwise comparison of all pairs of sequences, for example by pairwise alignment. During progressive alignment, the sequences are added one at a time to the already constructed subalignment. A guide tree, which is constructed from the distance matrix, controls in which order the sequences are added. In this scheme, it is crucial for the alignment quality that the most similar sequences are aligned first. A subsequent realignment step further refines the multiple alignment. Implementations like ClustalW [THG94], T-Coffee [NHH00], MAFFT [KMKM02] and MUSCLE [Edg04] differ in the techniques they use for the single steps and which additional methods they apply to improve the accuracy of the multiple alignment but all employ the progressive strategy.

2.3.2 Sequence-structure alignment

For non-coding RNAs (ncRNAs), sequence information alone is not sufficient to guide the alignment as the structure is usually more conserved than the sequence [TSH$^+$06]. So even if two sequences do not have a high sequence conservation, i.e. they do not share long stretches of the same nucleotides, they can still be related and form the same structure. In this case, the structure is conserved by *compensatory mutations* where a mutation in one end of a base pair is compensated by a mutation in the other end such that the base pair is preserved. A sequence-structure alignment computes the alignment that maximizes a score that incorporates both sequence *and* structure information.

Definition 2.7 (Sequence-structure alignment)
A *sequence-structure alignment* is the triple $(\mathcal{A}, R, T)$, where $\mathcal{A}$ is a sequence alignment (cf. Definition 2.6) and R and T are structures for sequences A and B, respectively. A base pair $(i, j) \in R$ $[(i, j) \in T]$ is *deleted* [*inserted*] by $(\mathcal{A}, R, T)$ iff positions i and j are *deleted* [*inserted*]. A base pair $(i, j) \in R$ $[(i, j) \in T]$ is *covered by* $(\mathcal{A}, R, T)$ iff there is a base pair $(i', j') \in T$ $[(i', j') \in R]$, such that $(i, i'), (j, j') \in \mathcal{A}$. Two base pairs $(i, j) \in R$ and $(i', j') \in T$ are *matched to each other by* $(\mathcal{A}, R, T)$ and form a *base pair match* iff $\mathcal{A}$ matches their respective left and right ends to each other.

Figure 2.3 displays a possible representation of a sequence-structure alignment. The plot is produced by RNAalifold [HFS02, BHW$^+$08] that is implemented in the ViennaRNA Package [HFS$^+$94, LBHZS$^+$11]. The alignment is annotated with the consensus structure that can be formed by most of the sequences and a color code that illustrates the extend of compensatory mutations for each base pair in the consensus structure. Furthermore, the sequence conservation is shown by gray bars for all alignment columns.

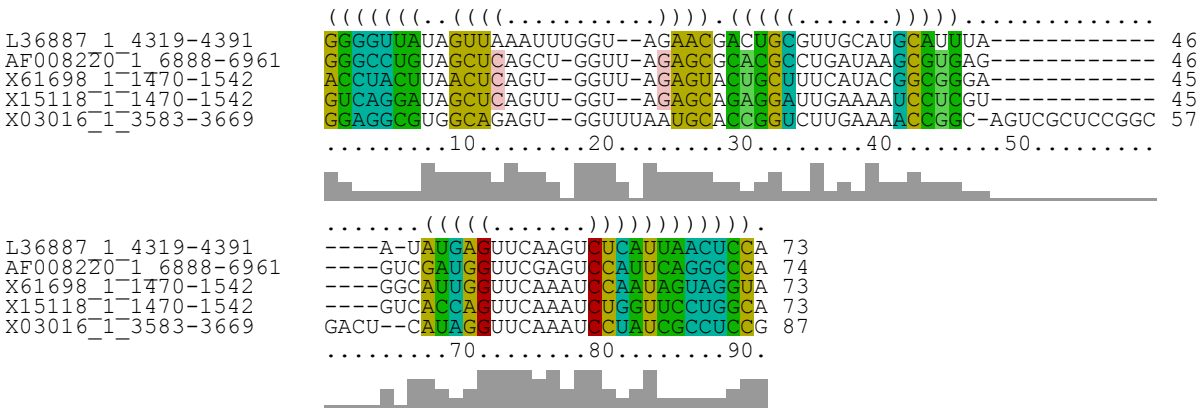


Figure 2.3: **Example sequence-structure alignment.** The alignment is produced for sequences from the tRNA family by the tool RNAalifold [LBHZS+11]. The consensus structure displayed at the top shows the characteristic cloverleaf structure of tRNA sequences. The sequence conservation of the different alignment columns is indicated by gray bars. The colors encode the number of types of base pairs that occur within one alignment column. Between the two red alignment columns, no compensatory mutations occur as all sequences consistently form a G−C base pair. On the other hand, 4 different base pairs occur in the turquoise alignment columns.

2.4 Overview of sequence-structure alignment methods

All sequence-structure alignment approaches can be categorized into the following three main groups: 1) structure information is inferred from a precomputed sequence alignment, 2) an alignment is computed based on the precomputed folded structures and 3) the sequences are simultaneously aligned and folded. These three variants are illustrated in Figure 2.4. In the following, a basic description of the different approaches including an overview of the available tools is given. A previous comparison was conducted in [GG04].

2.4.1 First align then fold

The first strategy applies a two step approach: first the input sequences are aligned using a sequence alignment tool, then a common secondary structure is derived from the (multiple) alignment. This *consensus structure* includes all base pairs shared by the majority of the sequences. Structures for each input sequence can be inferred by using the consensus structure as a constraint for folding (e.g. by RNAfold -C). In the following, a listing of the most common tools that apply the first strategy is given.

- RNAalifold [HFS02] is an extension of Zuker's algorithm [ZS81] that computes the minimum free energy structure for a given sequence alignment by averaging the energy contributions of all sequences in the alignment. Furthermore, it includes a covariation score to consider evolutionary relationships. Compensatory mutations in two alignment columns

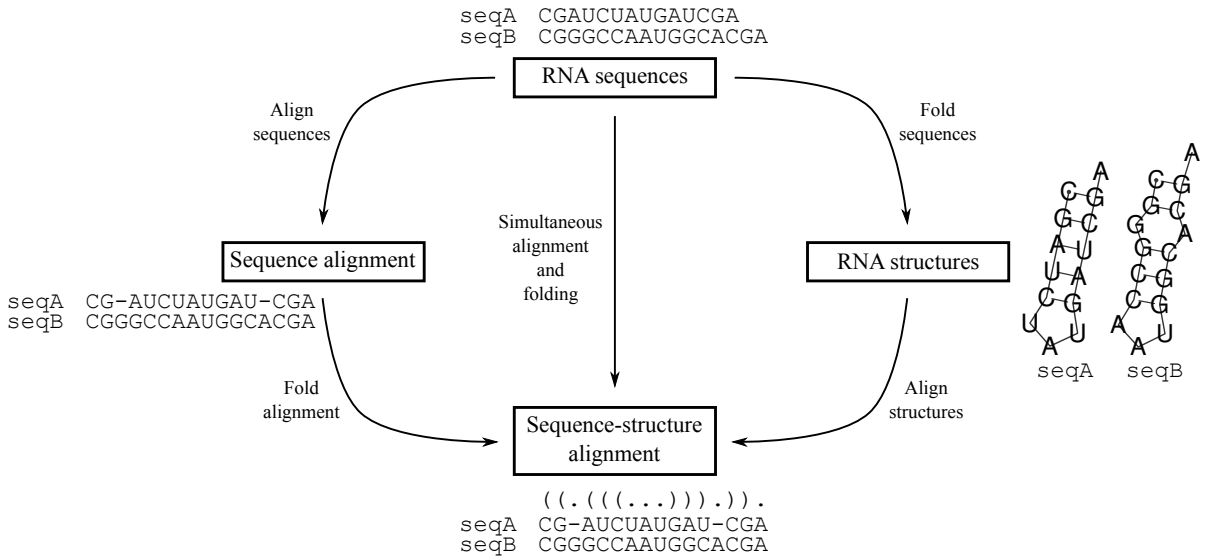


Figure 2.4: **Three variants for computing a sequence-structure alignment.** The first strategy first aligns the sequences and then folds the (multiple) alignment (left). The second approach first folds the sequences and then aligns the structures (right). The third variant folds and aligns the sequences simultaneously (middle). Figure adapted from [GG04], published under the Creative Commons Attribution License (`http://creativecommons.org/licenses/by/2.0`).

indicate the formation of a base pair between these positions. Thus, compensatory mutations are rewarded a base pair conservation score, whereas a penalty is considered for each sequence that cannot realize the base pair. The latter ensures that the consensus structure can be formed in most of the sequences. Both measures contribute to the covariation score. An advanced version of RNAalifold enhances the scoring and improves the handling of gaps [BHW+08].

- Pfold [KH99] models RNA secondary structure by a stochastic context free grammar (SCFG), where each production rule is assigned a probability. The SCFG is derived from a training set of correctly folded RNAs. Furthermore, the model includes mutation rates for single bases and base pairs. The evolutionary relationship of the input sequences is incorporated by the use of a phylogenetic tree that is computed by maximum likelihood estimation. The most likely secondary structure for a given sequence alignment is identified by maximum a posteriori estimation. A more accurate and faster version of Pfold is described in [KH03].

- ILM [RSZ04] extends the previous approaches by inferring pseudoknotted structures. Similar to RNAalifold, it applies a score that combines thermodynamic and covariation information.

All approaches that fold a precomputed (multiple) sequence alignment rely on the quality of that input alignment. Since sequence conservation is low in many RNA families, an alignment that is based on sequence information alone might not be accurate. Inferring covariation signals from severely misaligned sequences is not meaningful. Thus, for sequences with low sequence conservation, the consensus structure prediction often fails.

2.4.2 First fold then align

The second strategy reverses the order of the previous method: the sequences are first folded and then the resulting structures are aligned. In [Tai79], the concept of edit operations was generalized to trees. A tree alignment model was proposed in [JWZ95] as a similarity measure between two trees. In the following, a listing of the most common tools for the second strategy is given.

- RNAforester [HTGK03] extends the tree alignment algorithm in [JWZ95] with the ability to compute local forest alignments in the proposed forest representation of RNA structures. Thereby, local structural features common to given structures can be identified. A progressive profile approach for the extension of RNAforester for multiple structure alignment was introduced in [HVG04].
- MARNA [SB05] uses a pairwise alignment algorithm [JLMZ02] that minimizes the edit distance between two sequences that are annotated with a secondary structure. The model includes edit operations on bases and base pairs. Based on the pairwise alignments between all sequence pairs, T-Coffee's consistency transformation [NHH00] is conducted to favor those alignment edges that can be realized consistently between many of the sequences. The progressive alignment strategy of T-Coffee is used to construct a multiple alignment. Although technically only a sequence alignment is produced, it is guided by sequence *and* structure information that determines the weights of the alignment edges. A consensus structure is derived by measuring the correlation between the alignment columns.

Even though the presented methods for the second strategy are more robust towards low sequence conservation, using a single fixed structure for each sequence might pose problems. An RNA sequence might have several structures with a free energy close to the mfe structure. In such a case, it is not guaranteed that the approaches that first fix the structure of each input sequence and then align the structures find a proper consensus structure. Although MARNA can include information of several suboptimal structures, the computational complexity increases with the square of the number of considered structures. Thus, in practice, it is only feasible to consider a small number of suboptimal structures, which might not suffice for an accurate result.

2.4.3 Simultaneous alignment and folding

It was observed for the first two strategies that structure and alignment computation cannot be separated without potentially affecting the prediction quality. This is why a third variant performs the alignment and structure prediction *simultaneously*. Sankoff's algorithm [San85] is considered the gold standard even though it was proposed 30 years ago and has a high $O(n^6)$ time and $O(n^4)$ space complexity for the pairwise case. In [San85], it was discussed as a multiple sequence-structure alignment pipeline with extreme $O(n^{3N})$ time and $O(n^{2N})$ space complexity, where N denotes the number of sequences. Thus, various attempts have been made to obtain efficient implementations of restricted variants of the original Sankoff algorithm, which will be described in the following.

FoldAlign [GHS97] is designed to construct multiple local alignments by applying the pairwise progressive alignment strategy from ClustalW [THG94] while many suboptimal intermediate solutions are kept to increase the probability that the optimal solution is found. To reduce the computational complexity, only the number and not the energy of base pairs contribute to the structural score and multiloops cannot be predicted. The goal is to identify the "most significant core structure" [GHS97], which can be refined later by existent methods. This approach reduces Sankoff's time complexity to $O(N^4n^4)$. The approach was completely redesigned over the years to also include multiloops and a loop-based energy model [HLSG05] and efficient pruning of the dynamic programming matrix [HTG07].

Dynalign [MT02] is a pairwise sequence-structure alignment algorithm that restricts the maximal difference of positions between aligned nucleotides to a fixed parameter M and scores the sequence contribution only implicitly through free energy parameters. The time and space complexity is thereby reduced to $O(M^3n^3)$ and $O(M^2n^2)$, respectively.

A prominent line of research is based on constraints derived from sequence alignments, termed *sequence alignment-based* or *sequence-based* constraints. The main idea is to restrict the number of matrix cells through the identification of probable regions in the dynamic programming matrix based on pure sequence alignment. Only those cells that pass a fixed threshold are considered in the sequence-structure alignment. Stemloc [Hol05] provides a general model how to incorporate constraints into algorithms for pairwise SCFGs (pair SCFGs), which are utilized for evolutionary analysis. A pair SCFG extends the normal SCFG by emitting pairs of characters. Suboptimal alignments and foldings can be used as constraints to determine the "alignment and fold envelope" [Hol05]. These envelopes determine those parts of the DP matrix that are considered in the subsequent sequence-structure alignment computation based on a pair SCFG. Consan [DE06] also uses a pair SCFG but identifies confidently aligned position pairs that can guide the alignment process. These "pins" [DE06] are characterized by high posterior probabilities that are derived from pure sequence alignment. This speeds up the computation as only a reduced number of index combinations have to be considered in

a subsequent sequence-structure alignment. A revised version of Dynalign [HSM07] integrates probabilistic alignment constraints similar to those introduced in [Hol05, DE06]. Here, a hidden markov model (HMM) is utilized as a stochastic model to represent the sequence alignment and to derive "posterior co-incidence probabilities" [HSM07] for nucleotide pairs. For instances with low sequence conservation, however, all approaches that utilize sequence-based constraints have to cope with the fact that sequence information alone is not sufficient for sequence conservation under 60% [GWW05]. Either probable regions identified from pure sequence alignment might not be reliable or only small or no regions at all are discarded in the dynamic programming matrices. In the first case, the subsequent sequence-structure alignment might not be correctly predicted whereas the latter case would nullify the runtime advantage of those methods. In general, the runtime of the above mentioned tools that utilize sequence-alignment based constraints is between $O(n^3)$ and $O(n^6)$ for pairwise alignment.

Due to these shortcomings, a completely different route has been proposed with the tool PMcomp [HBS04]. To speed up the original Sankoff algorithm, the original free energy contribution in the loop-based energy model [MSZT99] is approximated by a product of base pair probabilities that can be efficiently precomputed by McCaskill's algorithm. This *lightweight* or base pair based energy model is further simplified by requiring that all predicted base pairs are covered by the sequence-structure alignment. The unconstrained version has $O(n^6)$ time and $O(n^4)$ space complexity, which matches the original Sankoff algorithm for the pairwise case. Restricted versions that limit the difference in size of matched base pairs reduce the runtime up to $O(n^4)$. The multiple alignment version PMmulti applies a progressive strategy and needs $O(N^2n^6)$ time for the unconstrained version.

LocARNA [WRH+07] is built upon PMcomp's simplified lightweight energy model, but further speeds up the computation by applying a fixed base pair filter. Base pairs with low probability within the structure ensemble are unlikely to contribute to an optimal solution and can be safely discarded without sacrificing the quality of the alignment. Thereby, only a linear number of base pairs need to be considered for each sequence, which reduces the runtime to $O(n^4)$ and the space complexity to $O(n^2)$ for the pairwise case. LocARNA's multiple alignment pipeline requires $O(N^2n^4)$ time. FoldAlignM [THG07] is a PMcomp variant that adds the ability to cluster the RNA sequences.

RAF [DFB08] is a hybrid that combines the lightweight energy model with sequence-based constraints. RAF inherits PMcomp's simplified structure prediction and applies fixed cutoffs to the base pair probabilities calculated by CONTRAfold [DWB06] and to the posterior probabilities of aligned positions computed by CONTRAlign [DGB06]. The combination of both approaches significantly reduces the runtime to $O(n^2)$ for pairwise alignment but also introduces the disadvantages of sequence-based constraints described above. RAF also includes a pipeline for computing multiple progressive alignments.

Recently, sparsification was also applied to the simultaneous alignment and folding prob-

lem [ZUGVWS08, ZUGVWS10], which reduces the time complexity over Sankoff's algorithm. However, the high time complexity still limits the practicability of this single non-heuristic optimization strategy.

All these considerations show that a lot of progress was made since Sankoff's algorithm was proposed in 1985. Nevertheless, all previous approaches are either too slow for large scale analyses or compromise the alignment quality for instances with low sequence conservation. But exactly these 'hard' alignment instances are ultimately decisive for assessing the quality of a structure alignment method. For this reason, novel fast approaches are required that perform well independent of the sequence conservation of the input sequences.

CHAPTER 3

Framework for ensemble-based sparsification

In this chapter, we introduce a novel form of sparsification, the *ensemble-based* sparsification. It is key to optimizing algorithms for the identification of sequence-structure motifs (cf. Chapter 4) and simultaneous alignment and folding (cf. Chapter 5). The sparsification technique introduced in [WZZU07] for RNA structure prediction is based on identifying parts of the computation that can be discarded without sacrificing optimality. Ensemble-based sparsification on the other hand determines those parts that are unlikely to contribute to an optimal solution and thus can reduce the time complexity much more. This chapter is based on [OMH+14].

The probabilities that a base or a base pair is contained in a loop (cf. Definition 3.1) are the basis for the ensemble-based sparsification. The main idea is to apply different filter steps to the structure ensemble. In the first filter step all base pairs are filtered by a fixed threshold. This was first proposed for the tool LocARNA [WRH+07]. We extend this technique by a second filter step, where all bases and base pairs within a loop are filtered based on the joint in-loop probabilities by applying two additional thresholds. Only those bases and base pairs that survived the filter can be – within the respective loop – part of the final result. Importantly, all these probabilities are efficiently precomputed independently for each sequence. Hence, e.g. in clustering scenarios, where all pairs from a set of sequences need to be matched, this preprocessing needs to be done only once for each sequence and not for all quadratically many pairs. We describe the procedure for sequence A of length n.

Definition 3.1 (Joint in-loop probabilities)
We define joint occurrence probabilities of elements in loops of structures in the Boltzmann ensemble of an RNA sequence A.

- $\Pr[k \in \text{loop}(i,j)|A]$ denotes for $i < k < j$ the joint probability that a structure of A contains the base pair (i,j) *and* the unpaired base k such that (i,j) is the parent of k.

- $\Pr[(i',j') \in \text{loop}(i,j)|A]$ denotes for $i < i' < j' < j$ the joint probability that a structure of A contains the base pairs (i,j) and (i',j') *and* that (i,j) is the parent of (i',j').

To simplify the notation, the expressions $\text{loop}(i,j)$ in Definition 3.1 resemble $\text{loop}_R(i,j)$ from Definition 2.5b) – notationally omitting the structures R in the Boltzmann ensemble of A. Based on these in-loop probabilities, we reduce the number of positions in the dynamic programming matrices that need to be filled.

Furthermore, we define external probabilities (see Definition 3.1), i.e. the probabilities that an unpaired base k or a base pair (i,j) is external and thus in the loop closed by the pseudo base pair $\psi_A = (0, n+1)$. As the pseudo base pair is present in each structure, we can utilize the notation of the joint in-loop probabilities. The external probabilities can be easily computed from the McCaskill matrices (cf. Section 2.2.3) as no base pair spans position k or base pair (i,j) and thus the structure can be partitioned in an arbitrary part before ($Q(1,k-1)$ and $Q(1,i-1)$) and after ($Q(k+1,n)$ and $Q(j+1,n)$) the base and base pair.

$$\begin{aligned} \Pr[k \in \text{loop}(0,n+1)|A] &= \frac{Q(1,k-1)Q(k+1,n)}{Q(1,n)} \\ \Pr[(i,j) \in \text{loop}(0,n+1)|A] &= \frac{Q(1,i-1)Q^b(i,j)Q(j+1,n)}{Q(1,n)} \end{aligned} \tag{3.1}$$

In Section 3.1 we describe how the base pairs per sequence can be reduced to a linear number. Details of how the joint in-loop probabilities from Definition 3.1 are computed can be found in Section 3.2 and 3.3. A complexity analysis is given in Section 3.4. In Section 3.5 we show how to utilize the joint in-loop probabilities to enable ensemble-based sparsification.

3.1 Restricting the number of base pairs

By considering only probable base pairs from both structure ensembles, the overall number of base pairs is reduced. This is achieved by introducing a fixed threshold θ_1 with which all base pairs are filtered, i.e. all base pairs (i,j) with $\Pr[(i,j)|A] \geq \theta_1$ are kept. In this way, only a linear number of base pairs remain (the argument has been given before in the context of LocARNA [WRH+07]). Furthermore, the threshold can be chosen to be suitable for the particular application.

Proposition 3.1 (Linear number of base pairs) *For a fixed $\theta_1 > 0$, there are only $O(n)$ base pairs with $\Pr[(i,j)|A] \geq \theta_1$.*

Proof. Because each structure of A has at most one base pair with right end j, it holds $\sum_i \Pr[(i,j)|A] \leq 1$. With this property, we can infer a bound on the number of base pairs.

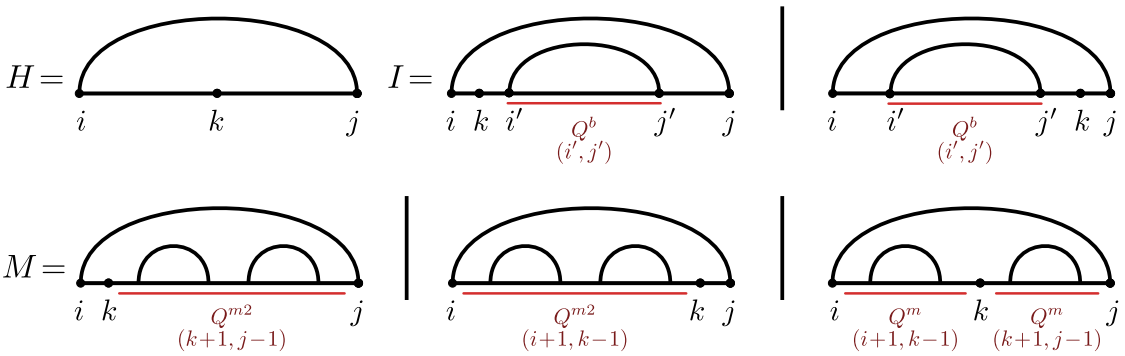


Figure 3.1: **Computation of unpaired probabilities in loops.** The recursions cover the cases that position k occurs unpaired inside a hairpin loop (H), 2-loop (I) or multiloop (M) with closing base pair (i, j). Every unpaired position in the multiloop is scored with $e^{-\mu c}$. Figure adapted from [OMH+14].

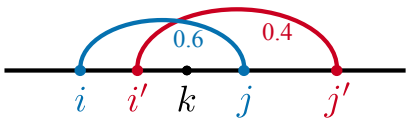
There exist at most $\frac{1}{\theta_1} \in O(1)$ base pairs with $\Pr[(i,j)|A] \geq \theta_1$ for a fixed position j. Thus, in total, there are at most $O(n)$ base pairs (i, j) with $\Pr[(i,j)|A] \geq \theta_1$. □

3.2 Computation of unpaired probabilities in loops

The unpaired probabilities in loops, i.e. $\Pr[k \in \text{loop}(i,j)|A]$, can be computed by extending McCaskill's algorithm. For this purpose, we utilize the matrices $Q(i,j)$, $Q^b(i,j)$, $Q^m(i,j)$ and $Q^{m1}(i,j)$ and energy terms as defined in Section 2.2.3. We extend the original set of matrices by introducing the additional matrix Q^{m2}.

$$Q^{m2}(i,j) = \sum_{i<k<j-1} Q^m(i,k) Q^{m1}(k+1,j)$$

It stores the partition function for parts of a multiloop with at least two interior base pairs for all subsequences $A_{i..j}$. We utilize the fact that matrix Q^{m1} always cuts at the left end of the right-most base pair to ensure that the recursions remain non-ambiguous. The joint probability that base pair (i, j) is contained in a structure and the unpaired base k occurs in the loop closed by (i, j), i.e. $\Pr[k \in \text{loop}(i,j)|A]$, can be computed by Equations 3.2a–f. Please compare the formulas with the visualization in Figure 3.1. The joint probability is split into the probability that base pair (i, j) occurs in the structure ensemble and the conditional probability that $k \in \text{loop}(i, j)$ under the condition that base pair (i, j) is part of the structure for sequence A, see Equation 3.2a. The conditional probability can be expressed by the partition function of all structures where k is unpaired in the loop closed by (i, j) divided by the partition function of all structures where i and j form a base pair. The numerator can be further partitioned according to the loop type (Equation 3.2b), i.e. k is unpaired inside a hairpin loop (H), 2-loop (I), or multiloop (M). Position k can be unpaired before or after the interior base pair of



$$\Pr[k \in \text{loop}(i,j)|A] = 0.6$$
$$\Pr[k \in \text{loop}(i',j')|A] = 0.4$$
$$\Pr[k \in \text{loop}(i,j)|(i,j) \wedge A] = 1$$
$$\Pr[k \in \text{loop}(i',j')|(i',j') \wedge A] = 1$$

Figure 3.2: **Joint vs. conditional probability.** Illustrative example for two structures that differ only in one base pair (shown in blue and red) with a respective probability of 0.6 and 0.4. Note that the blue and red base pair cannot be part of the same structure as the base pairs cross each other. Furthermore, there are no base pairs in the loop closed by (i,j) and (i',j'). This example illustrates that all joint probabilities of some parent base pair (blue and red base pair) and unpaired position k such that k is in the loop closed by the base pair sum up to at most 1 (here $0.6 + 0.4 = 1$), whereas the conditional probabilities do not necessarily fulfill this property (here $1 + 1 = 2$).

a 2-loop (Equation 3.2d) or, in the multiloop case, unpaired before, after (Equation 3.2e) or in between (Equation 3.2f) the interior base pairs of a multiloop. By utilizing matrix Q^{m2}, it can be assured that only proper multiloops that have at least two interior base pairs are considered. Precomputing the matrix allows to access a value of Q^{m2} in constant time. Note that the energy of a k-multiloop with u unpaired bases is calculated by $a + (k-1)b + uc$.

$$\Pr[k \in \text{loop}(i,j)|A] = \Pr[(i,j)|A] \Pr[k \in \text{loop}(i,j)|(i,j) \wedge A] \tag{3.2a}$$
$$= \Pr[(i,j)|A] \frac{H + I + M}{Q^b(i,j)}, \text{where} \tag{3.2b}$$

$$H = e^{-\mu E_H(i,j)} \tag{3.2c}$$
$$I = \sum_{\substack{i',j' \\ i<k<i'<j'<j}} e^{-\mu E_{SBI}(i,j,i',j')} Q^b(i',j') + \sum_{\substack{i',j' \\ i<i'<j'<k<j}} e^{-\mu E_{SBI}(i,j,i',j')} Q^b(i',j') \tag{3.2d}$$
$$M = e^{-\mu a} e^{-\mu c(k-i)} Q^{m2}(k+1, j-1) + e^{-\mu a} Q^{m2}(i+1, k-1) e^{-\mu c(j-k)} \tag{3.2e}$$
$$+ e^{-\mu a} Q^m(i+1, k-1) e^{-\mu c} Q^m(k+1, j-1) \tag{3.2f}$$

Now we have a closer look at the difference between joint and conditional probabilities in the given case and give a first intuition why we use joint probabilities for the novel ensemble-based sparsification. Unlike for conditional probabilities, all joint probabilities – that a structure of A contains the base pair (i,j) and k occurs as an unpaired base or right end of a base pair in the loop closed by (i,j) – sum up to at most 1 for a fixed position k and all parent base pairs; for an illustrative example see Figure 3.2. This is the property that we will utilize to obtain a low complexity. Similarly to the reasoning in Proposition 3.1, this is the essential key that guarantees that each position is considered only for a constant number of base pairs (the full proof will be given in Section 3.5).

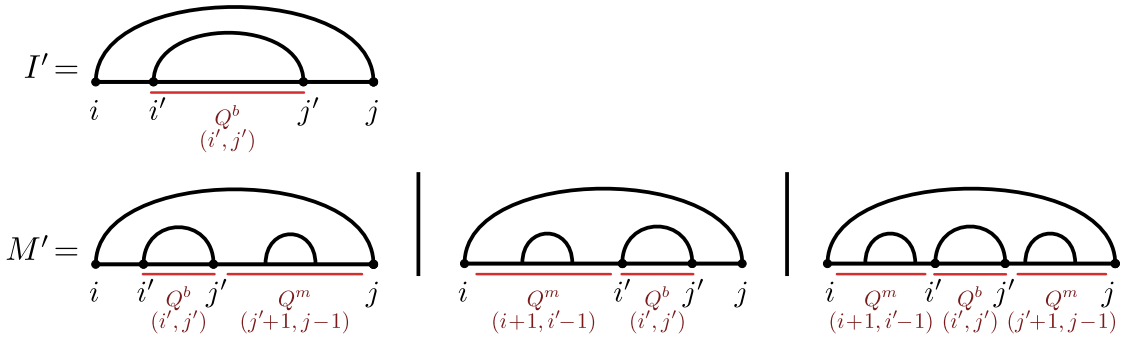


Figure 3.3: **Computation of base pair probabilities in loops.** The recursions cover the cases that base pair (i', j') occurs inside a 2-loop (I') or multiloop (M') with closing base pair (i, j). Every unpaired position in the multiloop is scored with $e^{-\mu c}$. Figure adapted from [OMH+14].

3.3 Computation of base pair probabilities in loops

The joint probability that base pair (i, j) is contained in a structure and the base pair (i', j') occurs in the loop closed by (i, j), i.e. $\Pr[(i', j') \in \text{loop}(i, j)|A]$, can be computed by Equations 3.3a–f. A visualization of the recursion can be found in Figure 3.3. Analogously to the computation of unpaired probabilities in loops, the conditional probability in Equation 3.3a is computed by considering the case that (i', j') is the interior base pair of a 2-loop (I') or multiloop (M'), see Equation 3.3b. In the multiloop case, base pair (i', j') can be the leftmost (Equation 3.3d), the rightmost (Equation 3.3e) or any other interior (Equation 3.3f) base pair. Note that the energy of a k-multiloop with u unpaired bases is calculated by $a + (k-1)b + uc$.

$$\Pr[(i', j') \in \text{loop}(i, j)|A] = \Pr[(i, j)|A] \Pr[(i', j') \in \text{loop}(i, j)|(i, j) \wedge A] \tag{3.3a}$$

$$= \Pr[(i, j)|A] \frac{I' + M'}{Q^b(i, j)}, \text{ where} \tag{3.3b}$$

$$I' = e^{-\mu E_{SBI}(i,j,i',j')} Q^b(i', j') \tag{3.3c}$$

$$M' = e^{-\mu a} e^{-\mu c(i'-i-1)} Q^b(i', j') Q^m(j' + 1, j - 1) \tag{3.3d}$$

$$+ e^{-\mu a} Q^m(i + 1, i' - 1) Q^b(i', j') e^{-\mu c(j-j'-1)} \tag{3.3e}$$

$$+ e^{-\mu a} Q^m(i + 1, i' - 1) Q^b(i', j') Q^m(j' + 1, j - 1) \tag{3.3f}$$

3.4 Complexity Analysis

The additional joint occurrence probabilities as defined in Definition 3.1 need to be computed only for the linear number of base pairs that remain after the filter step with threshold θ_1

(cf. Proposition 3.1). All time and space complexities in the following are given for this case. Matrix Q^{m2} can be precomputed in $O(n^2)$ time and space. For the computation of unpaired probabilities in loops, only $O(n^2)$ many combinations of (i, j) and k exist. Except the I recursion, each of them require constant time. By restricting the loop length or simplifying the energy calculation of 2-loops for the I recursion in the same way as in the McCaskill algorithm (see 2.2.3), the computation can be performed in at most linear time. For the computation of base pair probabilities in loops, $O(n^2)$ combinations of (i, j) and (i', j') exist, each of which can be computed in constant time. In total, all required joint in-loop probabilities can be computed in $O(n^3)$ time and $O(n^2)$ space, which is within the same complexity bounds as the McCaskill algorithm.

3.5 Implementing ensemble-based sparsification

Based on the introduced in-loop probabilities, we introduce now all necessary background to prepare for ensemble-based sparsification. This novel technique discards parts of the dynamic programming matrices that are unlikely to contribute to the optimal solution. This reduces the number of positions that need to be filled for the novel algorithms ExpaRNA-P and SPARSE (cf. Chapter 4 and 5). For that, we precompute all in-loop probabilities for both sequences A and B. Here, we show how to identify the relevant parts of the dynamic programming matrices and introduce additional data structures that allow an efficient traversal of the sparsified matrices.

In addition to θ_1 that is used as a cutoff for filtering the sets of base pairs, we introduce fixed thresholds θ_2 and θ_3 for the in-loop probabilities. These additional probability cutoffs are key for ensemble-based sparsification and drastically reduce the number of candidate positions that need to be considered.

Definition 3.2 (Candidate)
Each j' is a *candidate of* (i, j) *in sequence* A if it is either a significant single-stranded position within (i, j), i.e. $\Pr[j' \in \text{loop}(i, j)|A] \geq \theta_2$, or contained in a significant helix of (i, j), i.e. $\Pr[(i', j') \in \text{loop}(i, j)|A] \geq \theta_3$ for some i'. Analogously, we define *candidates* l' *of* (k, l) *in sequence* B: for candidates l' holds $\Pr[l' \in \text{loop}(k, l)|B] \geq \theta_2$ or $\Pr[(k', l') \in \text{loop}(k, l)|B] \geq \theta_3$ for some k'.

Due to the recursive structure of our novel algorithms, Definition 3.2 does not consider a position j' as a candidate if it is the left end of a probable base pair within the current loop. Centrally for the complexity of the novel algorithms ExpaRNA-P and SPARSE (cf. Chapter 4 and 5), Lemma 3.2 provides an upper bound for the number of times each position occurs as a candidate across all possible base pairs.

Lemma 3.2 *For a fixed j', there are only $O(1)$ base pairs (i,j), such that j' is a candidate of (i,j) (and analogously for l' and (k,l) in sequence B).*

Proof. We fix some j' and denote by $p_{j'}(i,j)$ the probability that a structure of A contains the base pair (i,j) and j' occurs as an unpaired base or right end of a base pair in the loop closed by the base pair (i,j):

$p_{j'}(i,j) := \Pr[j' \in \text{loop}(i,j)|A] + \sum_{i<i'<j'} \Pr[(i',j') \in \text{loop}(i,j)|A]$. If j' is a candidate, it follows $p_{j'}(i,j) \geq \theta^* := \min\{\theta_2, \theta_3\}$, since then either $\Pr[j' \in \text{loop}(i,j)|A] \geq \theta_2$ or $\Pr[(i',j') \in \text{loop}(i,j)|A] \geq \theta_3$ for some i'. Note that for different (i,j) the events of probabilities $p_{j'}(i,j)$ are disjoint, since in any structure j' can occur in just one loop. Therefore $\sum_{i,j} p_{j'}(i,j) \leq 1$. Hence there are at most $\frac{1}{\theta^*} \in O(1)$ base pairs (i,j) for which $p_{j'}(i,j) \geq \theta^*$ and j' can be a candidate only for those. □

By considering only candidate positions when computing the score of a base pair match, we can operate on sparsified dynamic programming matrices. In order to efficiently compute all necessary parts, additional data structures need to be introduced (cf. Equation 3.4). The vector $\mathsf{pos}^A_{(i,j)}$ contains all candidates of (i,j) in sequence A sorted in ascending order. The number of elements of vector $\mathsf{pos}^A_{(i,j)}$ is denoted by $\left|\mathsf{pos}^A_{(i,j)}\right|$. We introduce another vector $\mathsf{mat\text{-}idx\text{-}bef}^A_{(i,j)}$ that stores for each sequence position i' the first matrix index that lies before i'. The function mat-pos-bef$^{ijkl}(i',k')$, which is defined by $\mathsf{mat\text{-}idx\text{-}bef}$ in both sequences, allows an efficient traversal of the sparsified matrices.

$$\begin{aligned}
\mathsf{pos}^A_{(i,j)}(\bar{x}) &= j' \text{ where } j' \text{ is the } \bar{x}\text{-th candidate position in} \\
&\qquad \text{the loop closed by } (i,j) \text{ in } A \\
\mathsf{mat\text{-}idx\text{-}bef}^A_{(i,j)}(i') &= \max \bar{p} \text{ with } \mathsf{pos}^A_{(i,j)}(\bar{p}) < i', 0 \text{ if it does not exist} \\
\text{mat-pos-bef}^{ijkl}(i',k') &= (\mathsf{mat\text{-}idx\text{-}bef}^A_{(i,j)}(i'), \mathsf{mat\text{-}idx\text{-}bef}^B_{(k,l)}(k'))
\end{aligned} \tag{3.4}$$

The computation of these additional data structures does not increase the time complexity of $O(n^3)$ for precomputing all in-loop probabilities. This means that all preprocessing steps for ensemble-based sparsification can be done within the same complexity bounds as the McCaskill algorithm.

To make the description clearer, we distinguish sequence positions from matrix indices. We use $'$ ("prime") to denote a sequence position, e.g. i'. A position in the sparsified matrix is distinguished by using a $^-$("bar") and is denoted by $(\bar{x}, \bar{y})$. These notations will be consistently used in Chapter 4 and 5.

CHAPTER 4

Fast simultaneous exact pattern matching and folding

In this chapter, we introduce ExpaRNA-P – a novel approach for identifying sequence-structure motifs common to two RNAs. In contrast to already existent methods that solve the problem for *a priori known* structures, we consider exactly matching sequence-structure motifs in entire Boltzmann-distributed structure ensembles of two RNAs. This is important since reliable structures for single sequences are rarely known and single sequence structure prediction is usually highly unreliable. Furthermore, we present ExpLoc-P, a novel pipeline that utilizes the sequence-structure motifs identified by ExpaRNA-P to speed up simultaneous alignment and folding algorithms. Before we describe our novel approach in detail, we give a short overview of existent methods for detecting sequence-structure motifs. In addition, a detailed overview of methods for simultaneous alignment and folding is provided in Section 2.4, including LocARNA [WRH$^+$07] and Sankoff's algorithm [San85]. We describe all approaches for sequences A and B with respective lengths n and m ($n \geq m$) and associated base pair sets P^A and P^B. The description in this chapter is based on [OMH$^+$14]. A preliminary version was published in [SMH$^+$12].

Approaches for sequence-structure motif detection Various attempts have been made to efficiently find sequence-structure matches between two RNA sequences. The algorithm devised by Backofen and Siebert [BS07] efficiently computes common structure motifs that exactly match the underlying nucleotides for fixed secondary structures in $O(n^2)$ time and space. ExpaRNA [HWBB09] utilizes these motifs and identifies the best subset that can be simultaneously part of an alignment in $O(Hn^2)$ time and $O(n^2)$ space, where $H \in O(n^2)$ in general and $H \ll n^2$ for real RNAs. Note that this chaining approach guarantees that the motifs do not overlap or have crossing edges. The identified subset of motifs can be subsequently used

as alignment constraints (termed anchor constraints) to speed up the RNA alignment method LocARNA [WRH+07]. This complete pipeline is implemented in the tool ExpLoc [HWBB09]. A very similar heuristic pipeline was employed for the tool RNA-unchained [BCA14]. In contrast to ExpaRNA, the sequence-structure motifs that are identified in the first step have to be connected at sequence level and thus a complete structural motif can only be matched if the part under the base pairs can also be matched or it has to be partitioned into two separate motifs. For instance, the closing stem of a multiloop can only be matched as one motif if the whole multiloop and loop regions can be matched as well. If this is not possible, the left ends of the base pairs of the stem have to be matched independently from the right ends and two separate motifs are computed. A subsequent chaining algorithm computes the best anchor constraints in $O(k^2 \log k)$, where k is the number of motifs identified in the first step. These two approaches suffer from similar problems as the first generation of RNA alignment methods [HTGK03, SB05]: relying on a single predicted input structure for each sequence, this strategy fails frequently and causes severe misalignments, since predicting minimum free energy structures from single sequences is highly unreliable. On top of that, the motifs identified by RNA-unchained have to be connected on the sequence level such that base pairs cannot be matched in total without the loop region that connects the base pairs on the sequence level.

Contributions We present the novel algorithm ExpaRNA-P, which computes exactly sequence-structure-conserved elements that form highly probable local substructures in the RNA structure ensembles of both input RNAs. Analogous to Sankoff's simultaneous alignment and folding idea [San85], the novel strategy performs "simultaneous matching and folding" of RNA sequences. Thereby, it liberates exact pattern matching from its restriction to *a priori* fixed structures [BS07]. We point out that a straight-forward extension of the fixed input structure matching to simultaneous matching and folding would require at least $O(n^4)$ time and $O(n^2)$ space, which is still as high as the complexity of LocARNA. However, reducing this complexity is fundamental to speed up RNA comparison significantly.

Thus, our main technical contribution is to solve simultaneous matching and folding in quadratic time and space – as efficiently as plain sequence alignment. This is enabled by a novel ensemble-based sparsification technique (cf. Section 3) that substantially goes beyond prior approaches. Utilizing novel ensemble properties of the sequences, we identify sparse regions of each matrix such that, in total across all matrices, only quadratically many *matrix entries* have to be computed; each of them can be calculated in constant time. In contrast, LocARNA reduces only the number of computed *DP-matrices*, but requires quadratic time for each of them. This novel sparsification is based on limiting the joint probability of a sequence position or a base pair occurring as parts of particular loops in the ensembles of the single RNAs.

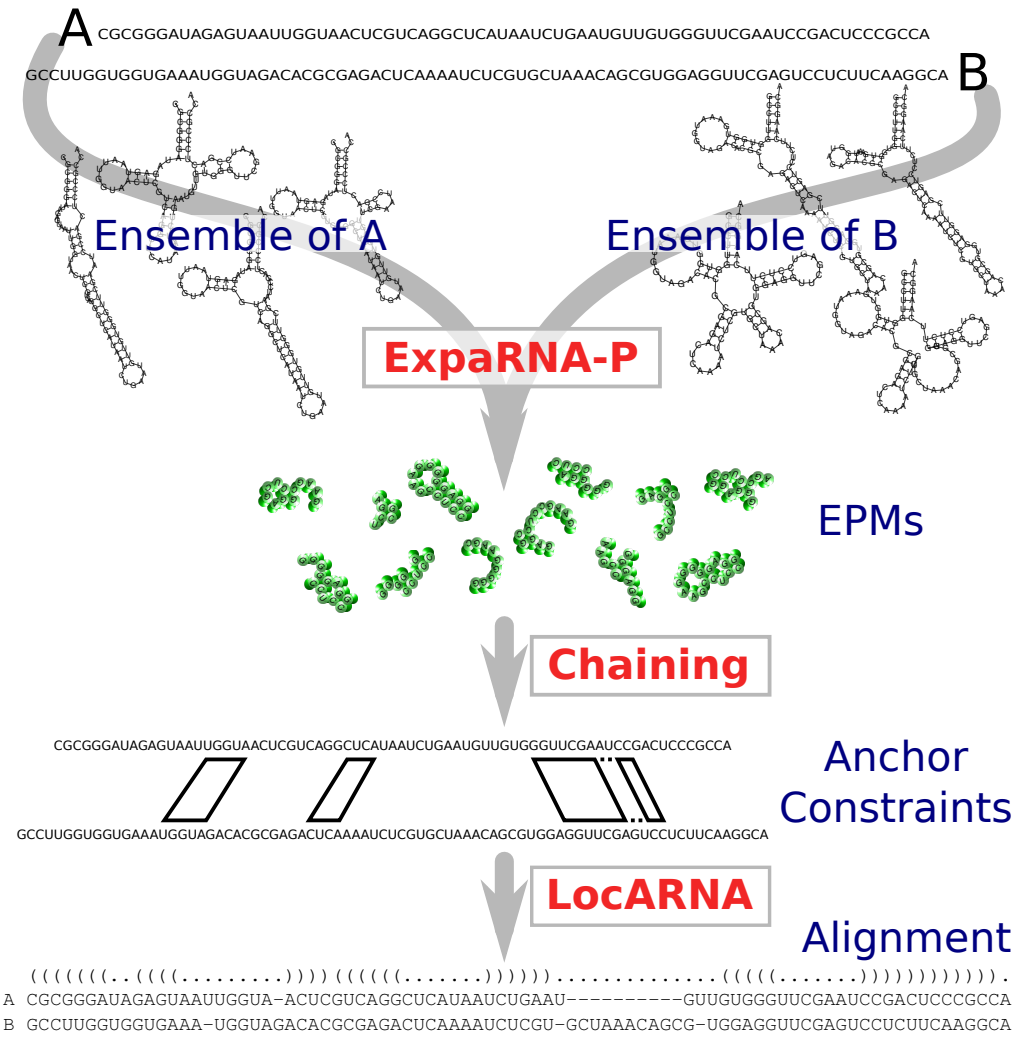


Figure 4.1: **Visualization of the ExpLoc-P pipeline.** ExpLoc-P uses exact pattern matchings as anchor constraints to speed up RNA structure alignments. 1) ExpaRNA-P identifies exact sequence-structure patterns (EPMs), 2) a chaining algorithm selects an optimal subset of compatible matchings that can simultaneously occur in an alignment of RNAs and 3) LocARNA utilizes these matchings as anchor constraints to speed up the alignment computation. Figure taken from [OMH+14].

Overview of Results To evaluate the practical benefits of our algorithmic innovations, we construct the pipeline ExpLoc-P for simultaneous alignment and folding (in the spirit of ExpLoc), which we sketch in Figure 4.1. In its first stage, it enumerates suboptimal exact matchings of local sequence-structure patterns with the novel algorithm ExpaRNA-P. In the second stage, the suboptimal matchings are chained to select an optimal subset of compatible matchings that can simultaneously occur in an alignment of RNAs. Finally, these matchings are heuristically utilized as anchor constraints in the subsequent LocARNA alignment. This procedure speeds up LocARNA as only the parts in between the anchor constraints have to be computed. We

show that ExpLoc-P produces high-quality alignments in extensive benchmarks. At the same time, due to its heuristic use of ExpaRNA-P anchors, it achieves a considerable speedup (about four-fold) over the benchmark set of typical RNAs (BRAliBase 2.1 [WMS06, GWW05]). For long sequences (≈400nt) of the benchmark set, the speedup is more than 30-fold.

Additionally, we study important design choices in the ExpLoc-P pipeline, which provides insights into practical implications of the developed concepts; in particular, we compare strict and relaxed matching in ExpaRNA-P. The latter allows mismatches at structural positions, which improves the coverage of low identity sequences.

4.1 ExpaRNA-P – Sparsifying the computation of pattern matchings

In the following, we introduce our novel algorithm ExpaRNA-P for simultaneous exact pattern matching and folding. We describe in detail which pattern matchings are considered and derive the recursion equations. To illustrate the different dynamic programming matrices, we first give the recursions for the unsparsified matrices. Afterwards we derive the proper recursions that operate on sparsified matrices.

4.1.1 Pattern matchings in RNA structure ensembles

ExpaRNA-P identifies sequence-structure patterns that are shared by two input RNA sequences. We provide a general description of pattern matchings in RNA sequences and specialize to two different variants (for examples, see Figure 4.2).

Definition 4.1 (Connected, Pattern Matching)
We denote the *match of positions i and k* by $(i{\sim}k)$ and the *base pair match of base pairs (i,j) and (k,l)* by $(ij{\sim}kl)$. We consider pairs $\mathcal{P} = (\mathcal{M}, \mathcal{S})$ of sets $\mathcal{M} \subseteq \{(i{\sim}k) | i \in [1..n], k \in [1..m]\}$ and $\mathcal{S} \subseteq \{(ij{\sim}kl) | (i,j) \in [1..n]^2, i < j, (k,l) \in [1..m]^2, k < l\}$.

$\mathcal{P}$ is *connected*, iff the graph $\mathcal{G}_\mathcal{P} = (\mathcal{M}, \mathcal{E})$, where $\mathcal{E} = \{((i{\sim}k),(j{\sim}l)) | (j = i+1 \text{ and } l = k+1) \text{ or } (ij{\sim}kl) \in \mathcal{S}\}$, is (weakly) connected.

$\mathcal{P}$ is called *Pattern Matching* iff

- $\mathcal{M}$ is a matching, i.e. $i = j \Leftrightarrow k = l$ for all $(i{\sim}k),(j{\sim}l) \in \mathcal{M}$
- $\mathcal{M}$ is non-crossing, i.e. $i < j \Rightarrow k < l$ for all $(i{\sim}k),(j{\sim}l) \in \mathcal{M}$
- $\mathcal{M}$ 'contains' $\mathcal{S}$, i.e. $(ij{\sim}kl) \in \mathcal{S} \Rightarrow (i{\sim}k),(j{\sim}l) \in \mathcal{M}$
- the structure $\{(i,j) | (ij{\sim}kl) \in \mathcal{S}\}$ is non-crossing (consequently, together with the previous condition, $\{(k,l) | (ij{\sim}kl) \in \mathcal{S}\}$ is non-crossing as well).

- $(\mathcal{M}, \mathcal{S})$ is connected.

A position i is *matched by* $\mathcal{P}$ *(in sequence* A*)* iff there is a position k, s.t. $(i \sim k) \in \mathcal{M}$. This is symmetrically defined for positions j and sequence B.

We are going to define strict and relaxed exact pattern matchings, see Figures 4.2a,b and Example 4.1 and introduce the term exact pattern matching (EPM) to refer to strict EPMs and relaxed EPMs generically. In the former, all matched nucleotides have to be identical. The latter relaxes this by allowing mismatched nucleotides at matched base pairs (taking

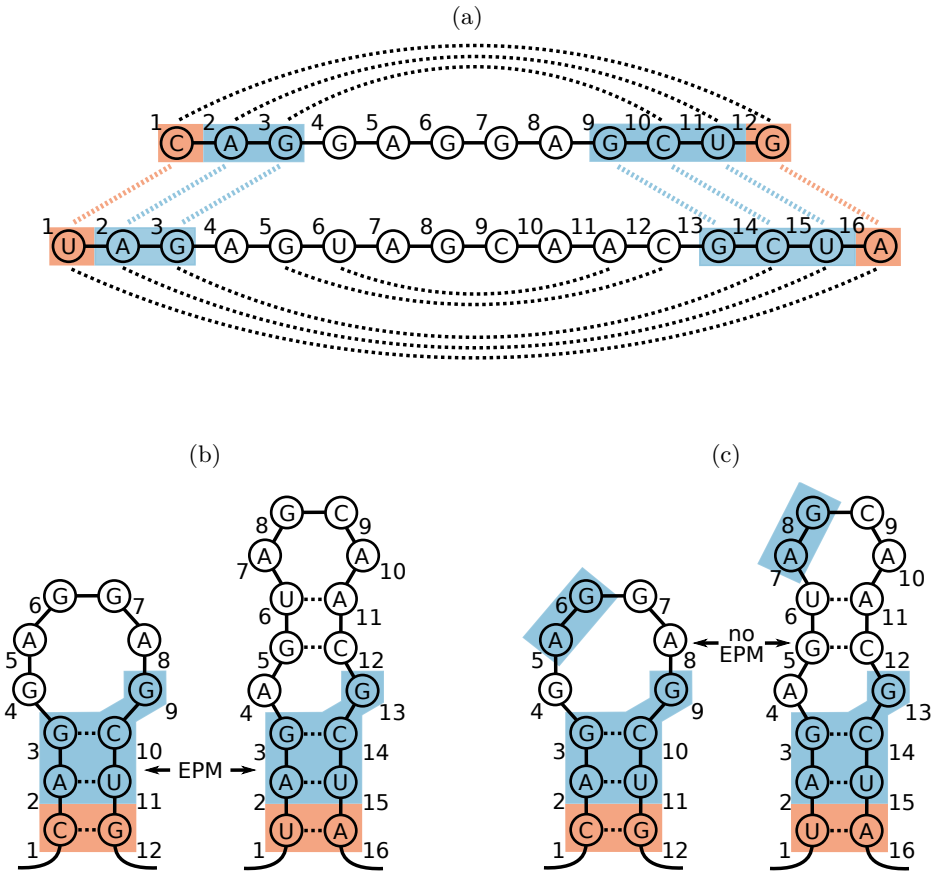


Figure 4.2: **Visualization of the pattern matching definition.** Exact matches are shown in blue and inexact (structure) matches in red. Two different illustrations of the notion pattern matching are shown in (a) and (b). The two separate regions in the blue parts are connected through base pairs and thus form a pattern matching, more precisely a strict EPM. The pattern matching can be extended by the (inexact) base pair match shown in red and forms a relaxed EPM. (c) shows an example of an invalid matching. Separately, both the small and the big matched parts are valid EPMs, but together they do not form a valid EPM as the two individual parts are not connected. Figure adapted from [OMH+14].

compensatory mutations into account).

For this purpose, we distinguish two kinds of matches in a pattern matching $(\mathcal{M}, \mathcal{S})$: define the set of structure matches as $\mathcal{M}|_{\mathcal{S}} := \{(i\sim k), (j\sim l) | (ij\sim kl) \in \mathcal{S}\}$; the set of sequence matches is $\mathcal{M} \setminus \mathcal{M}|_{\mathcal{S}} = \{(i\sim k) \in \mathcal{M} | (i\sim k) \notin \mathcal{M}|_{\mathcal{S}}\}$, i.e. all matches that are not structural matches.

Definition 4.2 (Strict EPM)
A *strict exact pattern matching (strict EPM)* is a pattern matching (Definition 4.1) with the additional property:

$$\text{for all } (i\sim k) \in \mathcal{M} : A_i = B_k.$$

Definition 4.3 (Relaxed EPM)
A *relaxed exact pattern matching (relaxed EPM)* is a pattern matching with the additional property:

$$\text{for all } (i\sim k) \in \mathcal{M} \setminus \mathcal{M}|_{\mathcal{S}} : A_i = B_k.$$

By Definition 4.1, a pattern matching, and therefore an EPM, does not necessarily match positions of contiguous subsequences, but it is required that the matched sequence-structure motifs are *structure-local* [BW04, OWB08] in each sequence. For example, in Figure 4.2b, the sets of blue sequence positions in each RNA are *structure-local*, because these positions are (graph-theoretically) connected via edges formed by backbone or base pair bonds; in contrast the blue motifs in Figure 4.2c are not structure-local, because they consist of two separated connected components.

Example 4.1 (Pattern matching definition)
We give a detailed example of the pattern matching definition for the sequences in Figure 4.2. Exact matches are shown in blue and inexact (structure) matches in red. For the EPM shown in blue, we obtain $\mathcal{M} = \{(2\sim 2), (3\sim 3), (9\sim 13), (10\sim 14), (11\sim 15)\}$ and $\mathcal{S} = \{(2\ 11\sim 2\ 15), (3\ 10\sim 3\ 14)\}$. Furthermore, the set of structure matches is given by $\mathcal{M}|_{\mathcal{S}} = \{(2\sim 2), (3\sim 3), (10\sim 14), (11\sim 15)\}$ and the set of sequence matches by $\mathcal{M} \setminus \mathcal{M}|_{\mathcal{S}} = \{(9\sim 13)\}$. As only exact matches occur, it is a strict EPM. The EPM can be extended by the base pair match shown in red, i.e. $\mathcal{M}' = \mathcal{M} \cup \{(1\sim 1), (12\sim 16)\}$ and $\mathcal{S}' = \mathcal{S} \cup \{(1\ 12\sim 1\ 16)\}$. The extended EPM is a relaxed EPM as mismatches in structure matches occur.

To characterize good EPMs, we define the *score of an EPM* $(\mathcal{M}, \mathcal{S})$ by summing up single score contributions of base and base pair matches:

$$\text{score}((\mathcal{M}, \mathcal{S})) = \sum_{(i\sim k)\in \mathcal{M}\setminus\mathcal{M}|_{\mathcal{S}}} \sigma(i,k) \ + \sum_{(ij\sim kl)\in\mathcal{S}} \tau(i,j,k,l), \tag{4.1}$$

where σ and τ are scoring functions with properties $\sigma(i,k) > 0$ if $A_i = B_k$ and $\tau(i,j,k,l) > 0$

if $A_i = B_k$ and $A_j = B_l$. The scoring parameters are instantiated by

$$\sigma(i,k) = \begin{cases} \alpha_1 & \text{if } A_i = B_k \\ -\infty & \text{otherwise} \end{cases}$$

$$\tau(i,j,k,l) = \alpha_1 \left(c_1(i,k) + c_1(j,l)\right) + \alpha_2 c_2(i,j,k,l) + \alpha_3 c_3(i,j,k,l)$$

$$c_1(i,k) = \begin{cases} 1 & \text{if } A_i = B_k \\ \text{str-mm} & \text{otherwise} \end{cases}$$

$$c_2(i,j,k,l) = \Pr\left[(i,j)|A\right] + \Pr\left[(k,l)|B\right]$$

$$c_3(i,j,k,l) = \Pr\left[(i,j) \wedge (i+1,j-1)|A\right] + \Pr\left[(k,l) \wedge (k+1,l-1)|B\right] \tag{4.2}$$

The parameters α_1, α_2, and α_3 weight respective contributions of matches $(i \sim k)$, and structure matches $c_2(i,j,k,l)$ and stacking $c_3(i,j,k,l)$ for a base pair match $(ij \sim kl)$. The stacking contribution $c_3(i,j,k,l)$ rewards stacked base pairs (i,j) and (k,l). Each mismatch at the left or right end of a base pair match is penalized by str-mm with str-mm < 0; for scoring strict EPMs, we set this penalty to $-\infty$, which forbids all kinds of mismatches. In analogy to the notation $\Pr\left[(i,j)|A\right]$, $\Pr\left[(i,j) \wedge (i+1,j-1)|A\right]$ denotes the joint probability of the stacked base pairs (i,j) and $(i+1,j-1)$. Such probabilities are computed in slight extension of McCaskill's algorithm [BBB+08].

As in the case of RNA structures (of some sequence A), one can define parent relations in EPMs of sequences A and B. In analogy, we define the *pseudo-base pair match* to match the two pseudo base pairs, i.e. $\psi := ((0,|A|+1) \sim (0,|B|+1)) = (\psi_A \sim \psi_B)$. In the following, we define an ordering of the base pair matches $(i'j' \sim k'l')$ by their spans $j'-i'+1$ (or $k'-l'+1$; the choice is arbitrary, since we consider only non-crossing structure). According to this partial order, we define $\text{parent}_{\mathcal{S}}(i \sim k)$ as the smallest $(i'j' \sim k'l') \in \mathcal{S} \cup \{\psi\}$ that satisfies $i' \leq i \leq j'$; $\text{parent}_{\mathcal{S}}(ij \sim kl)$ denotes the smallest base pair match that satisfies $i' < i < j < j'$.

For sequences $X \in \{A, B\}$, one efficiently computes base pair probabilities $\Pr\left[(i,j)|X\right]$ by McCaskill's algorithm [McC90]. Fundamentally, our novel sparsification technique relies on the joint probabilities $\Pr\left[k \in \text{loop}(i,j)|A\right]$ and $\Pr\left[(i',j') \in \text{loop}(i,j)|A\right]$ of Definition 3.1 that can be efficiently computed in the complexity bounds of the McCaskill algorithm (see Section 3.4). Since we want to match only structures that have high probability in the Boltzmann ensembles of the given sequences – as computed by McCaskill's algorithm [McC90] – we define the notion of significant EPMs. This constraint is crucial for both the quality of the results and the complexity of the algorithm. To define significance, we furthermore introduce three thresholds θ_1, θ_2 and θ_3. We limit the probability of all matched base pairs by θ_1; furthermore, the joint probabilities of matched unpaired bases and base pairs, occurring as part of their enclosing loop, by θ_2 and θ_3, respectively.

Definition 4.4 (Significant EPMs)
Given fixed thresholds θ_1, θ_2, and θ_3 (with $\theta_i > 0$ for $i \in \{1,2,3\}$), an EPM is *significant* iff

1. for all $(ij \sim kl) \in \mathcal{S}$: $\Pr[(i,j)|A] \geq \theta_1$ and $\Pr[(k,l)|B] \geq \theta_1$
2. for all $(j' \sim l') \in \mathcal{M} \setminus \mathcal{M}|_{\mathcal{S}}$ with $(ij \sim kl) = \text{parent}_{\mathcal{S}}(j' \sim l')$:
   - $\Pr[j' \in \text{loop}(i,j)|A] \geq \theta_2$ if $(i,j) \neq \psi_A$
   - $\Pr[l' \in \text{loop}(k,l)|B] \geq \theta_2$ if $(k,l) \neq \psi_B$
3. for all $(i'j' \sim k'l') \in \mathcal{S}$ with $(ij \sim kl) = \text{parent}_{\mathcal{S}}(i'j' \sim k'l')$:
   - $\Pr[(i',j') \in \text{loop}(i,j)|A] \geq \theta_3$ if $(i,j) \neq \psi_A$
   - $\Pr[(k',l') \in \text{loop}(k,l)|B] \geq \theta_3$ if $(k,l) \neq \psi_B$

We reduce the return set of our algorithm further by reporting only EPMs that are not included in better (reported) EPMs and that do not include better EPMs. Due to the negative scoring of mismatches within base pair matches, the second condition is relevant only for relaxed EPMs, since this cannot occur for strict EPMs. In the case of strict EPMs, those EPMs are simply *maximal* w.r.t. the following inclusion order $\sqsubseteq$ of pattern matchings. Hence, we call them *maximal strict EPMs*.

Definition 4.5 (Inclusion Order on EPMs)
Let $\mathcal{P} = (\mathcal{M}, \mathcal{S})$ and $\mathcal{P}' = (\mathcal{M}', \mathcal{S}')$ be EPMs. $\mathcal{P}$ is included in $\mathcal{P}'$, written $\mathcal{P} \sqsubseteq \mathcal{P}'$ iff

- $\mathcal{M} \subseteq \mathcal{M}'$
- for all $(i \sim k) \in \mathcal{M}$: $\text{parent}_{\mathcal{S}}(i \sim k) = \text{parent}_{\mathcal{S}'}(i \sim k)$[1]

Notably, in the inclusion order of Definition 4.5, EPMs with different structures are not comparable. Consequently, two EPMs that match the same positions can be both maximal, if they match different structure. In Figure 4.3a, both EPMs B and C are maximal.

In the case of strict EPMs, the highest scoring EPMs are always maximal EPMs w.r.t. the inclusion order, which allows us to select the "interesting" EPMs by this simple property. However, the same does not hold for relaxed EPMs: for example, typically the score of a relaxed EPM decreases if it is extended by a structure match with mismatching nucleotides; still, further extensions can increase the total score again. These dependencies are illustrated in Figure 4.3b: while extending EPM A, the score first decreases (EPM D) and then increases again (EPMs E and F).

Consequently, since we want to keep the highest scoring EPMs in the case of relaxed EPMs as well, we define a score-extended partial order.

[1] This condition is required to allow different structural variants (cf. Figure 4.3a) and to easily determine the starting points for the traceback (cf. Lemma 4.1).

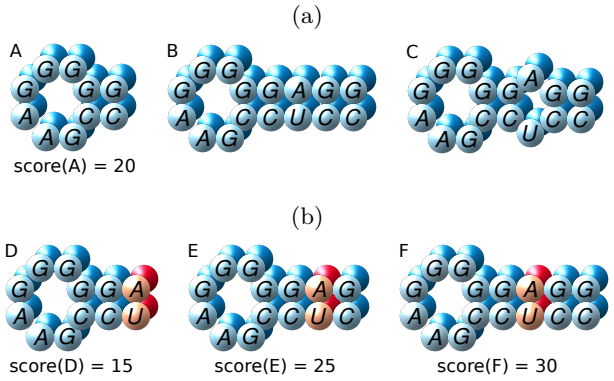


Figure 4.3: **Visualization of maximal EPMs.** Matches of blue bases refer to exact matches and red ones to inexact (structure) matches. (a) EPM A is not maximal since there exists a larger (strict) EPM (B or C). EPMs B and C can be maximal simultaneously since in each case some base matches have different parents. (b) EPM D is generated from A by appending an inexact structure match and has a lower score than A. Further extending the EPM leads to higher scores again (E and F). D is not maximal since A has the same parents and a higher score. A is not maximal because there exist (relaxed) EPMs E and F with the same parents and higher scores. Among A,D,E, and F, only F is maximal. Figure adapted from [OMH+14].

Definition 4.6 (Score Inclusion Order)
Let $\mathcal{P} = (\mathcal{M}, \mathcal{S})$ and $\mathcal{P}' = (\mathcal{M}', \mathcal{S}')$ be EPMs. $\mathcal{P}$ is smaller than $\mathcal{P}'$ in the score inclusion order, iff

- $\text{score}(\mathcal{P}) < \text{score}(\mathcal{P}')$
- $\mathcal{P} \sqsubseteq \mathcal{P}'$ or $\mathcal{P}' \sqsubseteq \mathcal{P}$

We call a relaxed EPM *maximal*, iff it is maximal w.r.t. this order among all relaxed EPMs. In other words, a relaxed EPM is maximal if and only if there is no second relaxed EPM with a higher score that is, by inclusion order, (a) smaller or (b) larger in the relaxed EPM. EPM D in Figure 4.3b for example is not maximal, as E and F are larger EPMs with the same parents that have a higher score. EPM F is maximal, as all other smaller EPMs have a lower score. Note that different patterns with the same score are not comparable so that they cannot rule out each other. Both maximality definitions are canonically raised to *maximal significant* strict EPMs and relaxed EPMs.

4.1.2 Optimizing over significant pattern matchings

Figure 4.4 provides formal recursion equations of the dynamic programming EPM optimization algorithm; the same recursions are presented graphically in Figure 4.5. The entries are recursively defined for all $(i, j) \in P^A$, $(k, l) \in P^B$, j' $(i < j' < j)$, and l' $(k < l' < l)$.

Fundamental to our approach, all matrices and evaluations in the recursions are sparse, i.e.

only entries and cases are considered where the probabilities of elements pass the respective probability thresholds (cf. Definition 4.4). Corresponding constraints are given in the recursion equations – this is also illustrated in Figure 4.5, using arrows. Otherwise, we can largely postpone this aspect until Section 4.1.3.

The matrix entries $D(ij, kl)$ score the best EPM enclosed by each base pair match $(ij \sim kl)$, i.e. $D(ij, kl)$ denotes the best score of a significant EPM $(\mathcal{M}, \mathcal{S})$ of $A_{i..j}$ and $B_{k..l}$ with $(ij \sim kl) \in \mathcal{S}$.

Inside the base pair match $(ij \sim kl)$, we determine the (score of the) best $(\mathcal{M}, \mathcal{S})$ that is either a significant EPM itself *or* forms a (connected) significant EPM only together with the closing base pair match $(ij \sim kl)$. The first case is covered by the single matrix L, whereas the latter case requires three matrices G_{A}, G_{AB}, and LR. By and large, for deriving one D-entry one starts matching from the left using L. Potentially, one introduces a gap using matrices G_{A} and G_{AB} and continues using matrix LR to match the part that will only be connected to the right end of $(ij \sim kl)$.

In more detail, first we determine the best score of a significant EPM $\mathcal{P} = (\mathcal{M}, \mathcal{S})$ that is connected to the left end $(i \sim k)$ of the base pair match, i.e. $\mathcal{M}$ is empty or contains $(i+1 \sim k+1)$. Concretely, $L^{ijkl}(j', l')$ is such a score, where $\mathcal{M} \subseteq [i+1..j'] \times [k+1..l']$ and $(j' \sim l') \in \mathcal{M}$. To introduce a gap, the latter condition is changed for G_{A} and G_{AB}. In the case of $G_{\mathrm{A}}^{ijkl}(j', l')$, $\mathcal{M}$ does not match j' but matches l'; for $G_{\mathrm{AB}}^{ijkl}(j', l')$, $\mathcal{M}$ does not match l' and potentially does not match j'. Finally, $LR^{ijkl}(j', l')$ is the best sum of scores of two significant EPMs $\mathcal{P}_1 = (\mathcal{M}_1, \mathcal{S}_1)$ and $\mathcal{P}_2 = (\mathcal{M}_2, \mathcal{S}_2)$ where the first is connected to the left base pair match end $(i \sim k)$ and the second contains $(j' \sim l')$. Intuitively, the two EPMs are separated by a gap; formally: (for all $(i_1 \sim k_1) \in \mathcal{M}_1$ and $(i_2 \sim k_2) \in \mathcal{M}_2$, $i_1 < i_2 - 1$ and $k_1 < k_2$) or (for all $(i_1 \sim k_1) \in \mathcal{M}_1$ and $(i_2 \sim k_2) \in \mathcal{M}_2$, $i_1 < i_2$ and $k_1 < k_2 - 1$).

Our recursion equations (Figure 4.4 and Figure 4.5) show the precise case distinctions and dependencies. In L, we check whether there is a sequence match (second case) or a structure match (third case); otherwise, we assign $-\infty$ (first case). To cover the structure match case, we iterate over all P^A and P^B, i.e. all possible base pairs for sequence A and B, that remain after filtering with θ_1 and θ_3 (cf. Definition 4.4 condition 1 and 3). LR is analogous to L, only allowing to close a gap left of the structure or sequence match. For this purpose, we introduce an auxiliary matrix H, which does not need to be stored.[2] The gap itself, computed in G_{A} and G_{AB}, allows skipping an arbitrary number of positions in both sequences. The recursion structure ensures that such a gap is introduced at most once per loop match and sequence. To avoid ambiguity, the recursion enforces to first skip positions in A (using G_{A}) and after that positions in B (using G_{AB}); furthermore we enforce a gap in the matchings computed via LR by its initialization.

[2] It is also possible to store the H matrix instead of the LR matrix. We chose this representation to emphasize the similarity between matrices L and LR.

$$D(ij,kl) = \max \begin{cases} -\infty \\ \text{if } \Pr[(i,j)|A] \geq \theta_1 \text{ and } \Pr[(k,l)|B] \geq \theta_1 \\ \quad \max\{L^{ijkl}(k-1,l-1), H^{ijkl}(k-1,l-1)\} + \tau(i,j,k,l) \end{cases}$$

$$L^{ijkl}(j',l') = \max \begin{cases} -\infty \\ \text{if } A_{j'} = B_{l'},\ \Pr[j' \in \text{loop}(i,j)|A] \geq \theta_2,\ \Pr[l' \in \text{loop}(k,l)|B] \geq \theta_2 \\ \quad L^{ijkl}(j'-1,l'-1) + \sigma(j',l') \\ \text{for all } (i',j') \in P^A, (k',l') \in P^B \\ \text{with } \begin{array}{l} \Pr[(i',j')|A] \geq \theta_1, \Pr[(i',j') \in \text{loop}(i,j)|A] \geq \theta_3 \text{ and} \\ \Pr[(k',l')|B] \geq \theta_1, \Pr[(k',l') \in \text{loop}(k,l)|B] \geq \theta_3 \end{array} \\ \quad L^{ijkl}(i'-1,k'-1) + D(i'j',k'l') \end{cases}$$

$$\begin{aligned} G_{\text{A}}^{ijkl}(j',l') &= \max\{L^{ijkl}(j'-1,l'), G_{\text{A}}^{ijkl}(j'-1,l')\} \\ G_{\text{AB}}^{ijkl}(j',l') &= \max\{L^{ijkl}(j',l'-1), G_{\text{A}}^{ijkl}(j',l'-1), G_{\text{AB}}^{ijkl}(j',l'-1)\} \\ H^{ijkl}(j',l') &= \max\{LR^{ijkl}(j',l'), G_{\text{A}}^{ijkl}(j',l'), G_{\text{AB}}^{ijkl}(j',l')\} \end{aligned}$$

$$LR^{ijkl}(j',l') = \max \begin{cases} -\infty \\ \text{if } A_{j'} = B_{l'},\ \Pr[j' \in \text{loop}(i,j)|A] \geq \theta_2,\ \Pr[l' \in \text{loop}(k,l)|B] \geq \theta_2 \\ \quad H^{ijkl}(j'-1,l'-1) + \sigma(j',l') \\ \text{for all } (i',j') \in P^A, (k',l') \in P^B \\ \text{with } \begin{array}{l} \Pr[(i',j')|A] \geq \theta_1, \Pr[(i',j') \in \text{loop}(i,j)|A] \geq \theta_3 \text{ and} \\ \Pr[(k',l')|B] \geq \theta_1, \Pr[(k',l') \in \text{loop}(k,l)|B] \geq \theta_3 \end{array} \\ \quad H^{ijkl}(i'-1,k'-1) + D(i'j',k'l') \end{cases}$$

$$F(j',l') = \max \begin{cases} 0 \\ \text{if } A_{j'} = B_{l'} \\ \quad F(j'-1,l'-1) + \sigma(j',l') \\ \text{for all } (i',j') \in P^A, (k',l') \in P^B \\ \text{with } \Pr[(i',j')|A] \geq \theta_1 \text{ and } \Pr[(k',l')|B] \geq \theta_1 \\ \quad F(i'-1,k'-1) + D(i'j',k'l') \end{cases}$$

Figure 4.4: **Recursion Equations.** Recursions for computing the significant strict EPMs and relaxed EPMs, respectively. These equations are visualized in Figure 4.5. Figure adapted from [OMH+14].

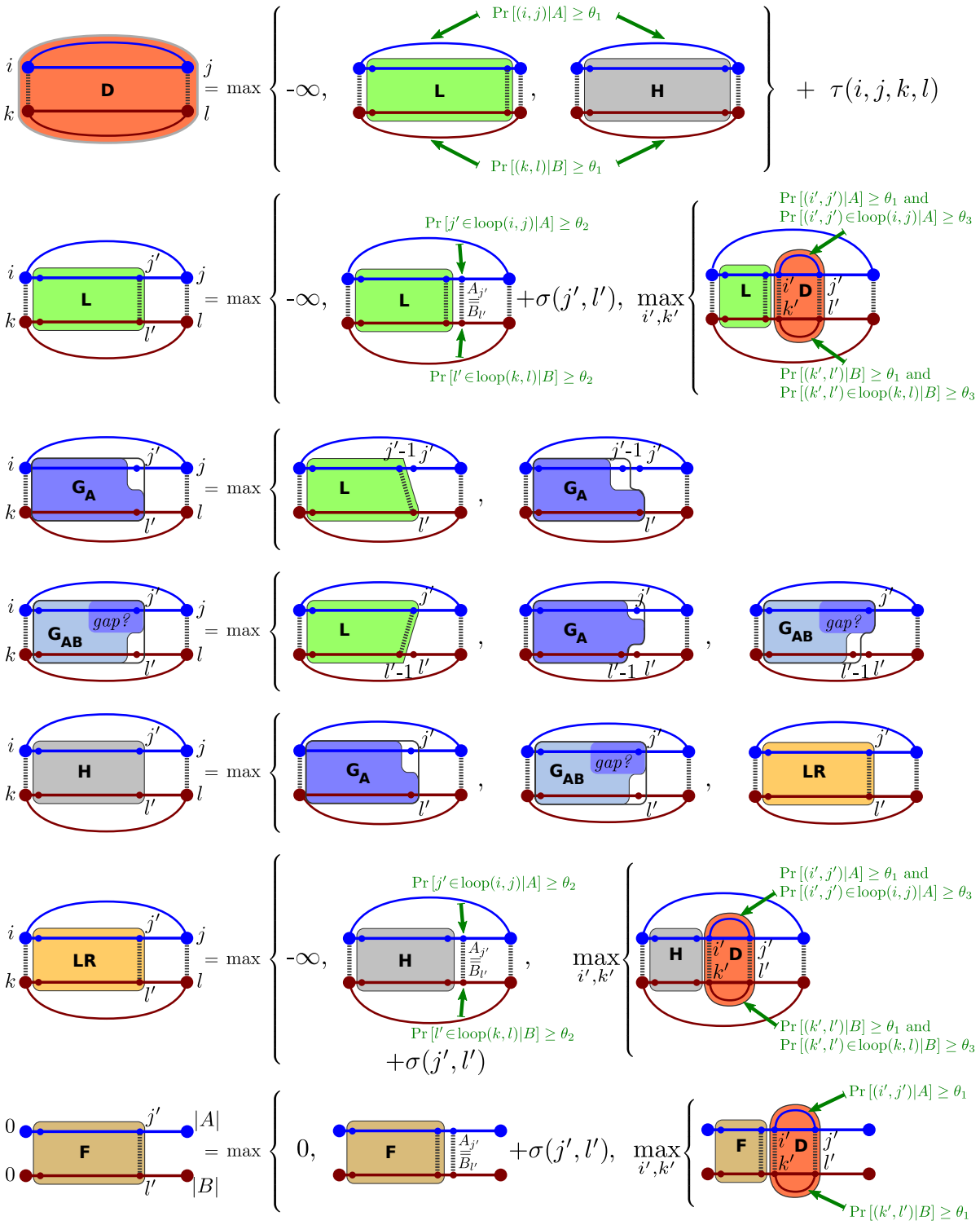


Figure 4.5: **Recursion Visualization.** Visualization of the recursions to compute the matrix entries $L^{ijkl}(j',l')$, $G_{\mathrm{A}}^{ijkl}(j',l')$, $G_{\mathrm{AB}}^{ijkl}(j',l')$, $LR^{ijkl}(j',l')$, $D(ij,kl)$, $F(j',l')$ and the auxiliary matrix $H^{ijkl}(j',l')$. Figure taken from [OMH+14].

We compute entries of D in increasing order with respect to their size so that when computing some $D(ij, kl)$, any $D(i'j', k'l')$ with $i < i' < j' < j$ and $k < k' < l' < l$ is already computed. Since EPMs are not necessarily closed by a base pair match (like the EPMs of D), we finally compute the matrix F. The entries $F(j', l')$, for $0 \leq j' \leq n$ and $0 \leq l' \leq m$, denote the maximum score of a significant EPM of $A_{1..j'}$ and $B_{1..l'}$ with $(j' \sim l') \in \mathcal{M}$. The recursion for F is almost identical to the recursion for L, except for the first case, which is 0 instead of $-\infty$, since the EPMs in F can start at any point (similar to local sequence alignments). Also, since the matched base pairs in EPMs of F are external (i.e. they are not enclosed by some other base pair of the EPM), we do not perform checks for the second and third condition of significant EPMs (Definition 4.4).

Matrix initialization Matrix entries corresponding to matches of empty subsequences are initialized. Here, we take special care to disallow such matches for certain matrices (by assigning $-\infty$).

- $L^{ijkl}(i,k) = G_{\mathrm{A}}^{ijkl}(i,k) = G_{\mathrm{AB}}^{ijkl}(i,k) = 0$ and $LR^{ijkl}(i,k) = -\infty$ (first matrix entry)
- $L^{ijkl}(i,l') = G_{\mathrm{A}}^{ijkl}(i,l') = LR^{ijkl}(i,l') = -\infty$ and $G_{\mathrm{AB}}^{ijkl}(i,l') = 0$ for all $l' > k$ (first matrix row)
- $L^{ijkl}(j',k) = G_{\mathrm{AB}}^{ijkl}(j',k) = LR^{ijkl}(j',k) = -\infty$ and $G_{\mathrm{A}}^{ijkl}(j',k) = 0$ for all $j' > i$ (first matrix column)

By initializing the LR matrix with $-\infty$, we keep matchings represented by LR and L distinct (because in this way, finite LR entries have to be derived via G_{A} or G_{AB} entries, which enforces a gap). The final matrix F is initialized by $F(j', 0) = F(0, l') = 0$ for all j', l'.

For enumerating only maximal EPMs during suboptimal traceback, we take special care that EPMs cannot be extended at the left or right end of gaps (G_{A} and G_{AB} matrices). For strict EPMs this is decided independently of the other traced strict EPMs. It suffices to check whether the strict EPM can be extended into the gap matrices, i.e. whether a sequence or structure match is possible at the borders of the gap matrices. However, the same does not work for relaxed EPMs, since while extending a relaxed EPM, the score might first decrease and then increase again (Figure 4.3). Therefore, we filter relaxed EPMs in two steps. First, we discard EPMs due to the same criterion as in the case of strict EPMs, checking for *exact* sequence or structure matches at the borders of the gap matrices. If an EPM cannot be discarded in this way, it is stored until all relaxed EPMs in the same D matrix are traced back. Only then, we compare the withheld relaxed EPMs of the same D matrix according to Definition 4.6. Since we complete the whole traceback for a D matrix before tracing into its "enclosed" D matrices, we identify and remove all non-maximal relaxed EPMs in an early stage of the traceback. To enumerate all maximal EPMs, we start such tracebacks only from entries $F(j', l')$ that satisfy $A_{j'+1} \neq B_{l'+1}$. Due to Lemma 4.1, this condition is necessary and sufficient for strict EPMs.

Lemma 4.1 *Let $\mathcal{P} = (\mathcal{M}, \mathcal{S})$ be a maximal strict EPM of $A_{1..j'}$ and $B_{1..l'}$ with $(j' \sim l') \in \mathcal{M}$. $\mathcal{P}$ is a maximal strict EPM of A and B, iff $A_{j'+1} \neq B_{l'+1}$.*

Proof. "⇒": Let $A_{j'+1} = B_{l'+1}$. Then $\mathcal{P}' := (\mathcal{M} \cup \{(j'+1 \sim l'+1)\}, \mathcal{S})$ is a strict EPM with $\mathcal{P} \sqsubseteq \mathcal{P}'$; hence $\mathcal{P}$ is not maximal for A and B (i.e. among all strict EPMs of A and B).
"⇐": Let $A_{j'+1} \neq B_{l'+1}$. Assume $\mathcal{P}$ is not maximal for A and B. Then, there is a strict EPM $\mathcal{P}' = (\mathcal{M}', \mathcal{S}') \neq \mathcal{P}$ with $\mathcal{P} \sqsubseteq \mathcal{P}'$ that is not a strict EPM of $A_{1..j'}$ and $B_{1..l'}$. Consequently, to satisfy $\mathcal{M} \subset \mathcal{M}'$, there has to exist $(ij \sim kl) \in \mathcal{S}'$ with $i \leq j' < j$ and $k \leq l' < l$. Consider first the case that $(j' \sim l') \in \mathcal{M} \setminus \mathcal{M}|_{\mathcal{S}}$. While the parent of $(j' \sim l')$ in $\mathcal{S}$ is ψ, there is a parent of $(j' \sim l')$ in $\mathcal{S}'$ different from ψ (i.e. either $(ij \sim kl)$ or some "smaller" base pair match). Let us now consider the case that $(j' \sim l') \in \mathcal{M}|_{\mathcal{S}}$. We know that $(i \sim k) \in \mathcal{M}$ and $\text{parent}_{\mathcal{S}}(i \sim k) \neq \text{parent}_{\mathcal{S}'}(i \sim k) = (ij \sim kl)$. Thus, this contradicts $\mathcal{P} \sqsubseteq \mathcal{P}'$ in both cases, because $\mathcal{P}$ and $\mathcal{P}'$ are not comparable by inclusion order (Definition 4.5). □

By the same argument, the forward direction holds for relaxed EPMs. Therefore, we enumerate all maximal relaxed EPMs by restricting the traceback in the same way. However, since the backward direction does not hold generally, this procedure can enumerate non-maximal relaxed EPMs. In practice, we observe this very rarely; consequently, while redundant relaxed EPMs could be removed explicitly, we let the chaining procedure handle those EPMs.

4.1.3 Recursions on sparsified matrices

In this section, we give a full description of the recursions operating on sparsified matrices. These matrices only contain relevant entries, i.e. those that can contribute to a significant EPM. If a position in one sequence is not likely to be unpaired or the right end of a base pair in the loop closed by a base pair of the base pair match, the position cannot be part of a significant EPM. Thus, the corresponding row or column in the matrix is no candidate (cf. Definition 3.2) and thus can be skipped. All necessary definitions for this novel ensemble-based sparsification are described in Chapter 3. A sequence position is denoted by a "prime", e.g. i', and a position in the sparsified matrix by a "bar", e.g. $(\bar{x}, \bar{y})$.

As we skip whole rows and columns in the sparsified matrix, the corresponding sequence positions of two adjacent matrix positions are not necessarily adjacent. In this case, an implicit gap is introduced. Consequently, the key modification extends the recursions to include the check of such a case. With the help of the data structures introduced in Definition 3.4, the check for adjacency wo-gap$^{ijkl}((\bar{x}, \bar{y}), (i', k'))$ (read "match without gap") can be computed in constant time:

$$\text{wo-gap}^{ijkl}((\bar{x}, \bar{y}), (i', k')) = [\mathsf{pos}^{A}_{(i,j)}(\bar{x}) - 1 = i' \wedge \mathsf{pos}^{B}_{(k,l)}(\bar{y}) - 1 = k'],$$

where $(\bar{x}, \bar{y})$ is a matrix position and i' and k' are sequence positions.

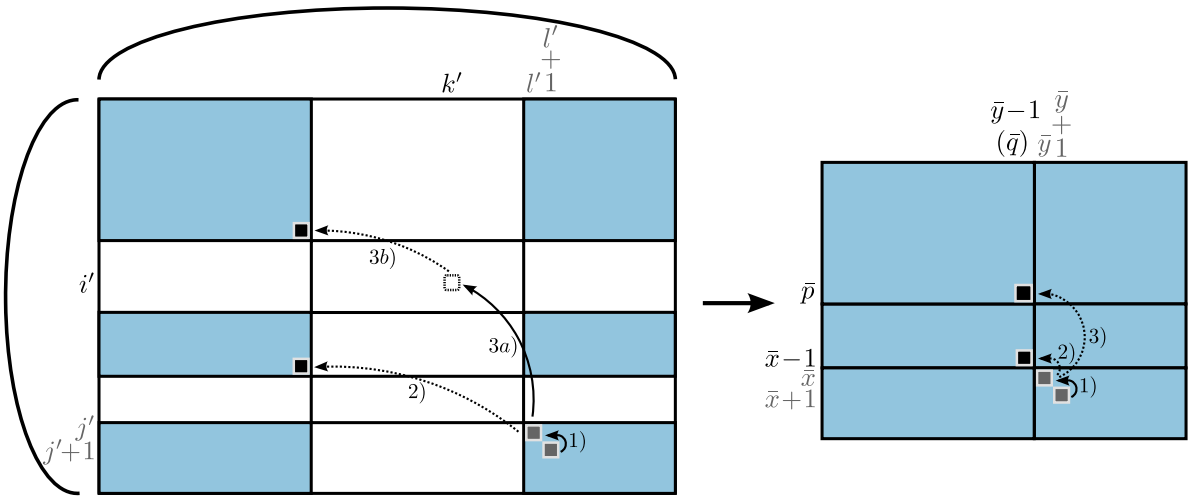


Figure 4.6: **Illustration of the recursion cases in the sparsified matrices.** The parts that correspond to candidates and thus need to be filled are shown in blue and the resulting sparsified matrix on the right side. The positions displayed in gray are in matrix LR; in black in matrix L, G_{A} or G_{AB}. After matching $\bar{x}+1$ and $\bar{y}+1$ in LR, we can continue matching from position $(\bar{x}, \bar{y})$ in LR as no gap between the corresponding sequence positions is introduced – shown with the solid arrow 1). The dotted arrows show the transition from matrix LR to matrix L, G_{A} or G_{AB}. This happens if an (implicit) gap is introduced as illustrated by 2) and 3*b*) since in each base pair match at most one gap can be introduced. For the structure match 3*a*) we go to the next valid matrix position $(\bar{p}, \bar{q})$ (cf. 3*b*)) that lies before the left ends i' and k' of the base pairs. A detailed description can be found in the text.

An illustration of the sparsified matrices is given in Figure 4.6. The blue parts of the matrices highlight candidates of the two enclosing base pairs of the base pair match (cf. Definition 3.2). During the computation, only these positions need to be calculated such that the recursions operate on sparsified matrices, see right matrix in Figure 4.6. The positions shown in gray belong to matrix LR. The positions shown in black are in matrix L, G_{A} or G_{AB}. After the first sequence match of $\bar{x}+1$ and $\bar{y}+1$ in matrix LR, we can continue matching in LR from position $(\bar{x}, \bar{y})$ – shown with the solid arrow 1) – as the corresponding sequence positions (l' and $l'+1$, j' and $j'+1$) are also directly adjacent and no gap is introduced. After the second sequence match of $\bar{x}$ and $\bar{y}$ in LR, we cannot continue matching in LR from position $(\bar{x}-1, \bar{y}-1)$ as the corresponding sequence positions are not adjacent and an implicit gap is introduced, see dotted arrow 2). In this case, the matching has to be continued in matrix L, G_{A} or G_{AB} as at most one gap is allowed for one base pair match. For the structure match ($i'j' \sim k'l'$), see arrow 3*a*), we go to the next valid matrix position that lies before the left ends i' and k' of the base pairs (dashed box). In the displayed example, a gap has to be introduced, see 3*b*), and the value at matrix position $(\bar{p}, \bar{q})$ in matrix L, G_{A} or G_{AB} is accessed. Note that, contrary to

the right ends of a base pair match, the left ends are not represented in the sparsified matrix (cf. candidate definition in Definition 3.2). Still, the base pair match can be included in the EPM and the next valid matrix position $(\bar{p}, \bar{q})$ is computed by accessing mat-pos-bef$^{ijkl}(i', k')$ (cf. Equation 3.4). Figure 4.7 shows the recursion on the sparsified matrices. In order to be able to check for adjacency within matrix H, we pass in addition the current sequence positions i' and k'.

ExpaRNA-P's efficiency depends fundamentally on the sparsity of the DP matrices, which we leverage through fixed thresholds θ_1, θ_2, and θ_3. We compute matrices L^{ijkl}, G_{A}^{ijkl}, G_{AB}^{ijkl}, and LR^{ijkl} only for base pairs (i, j) and (k, l) that are significant, i.e. $\Pr[(i, j)|A] \geq \theta_1$ and $\Pr[(k, l)|B] \geq \theta_1$. Furthermore, we compute only relevant entries of these matrices, namely those rows and columns whose index of the enclosing base pair as defined in Definition 3.2.

Before we give the full proof in Corollary 4.3, we give the key idea that leads to ExpaRNA-P's quadratic time complexity. By definition, only candidates j' or l' can be part of a significant EPM as defined in Definition 4.4; otherwise, we assign $-\infty$ to $L^{ijkl}(j', l')$ and $LR^{ijkl}(j', l')$. Furthermore, in the latter case, we neither store nor compute the values for $G_{\mathrm{A}}^{ijkl}(j', l')$ and $G_{\mathrm{AB}}^{ijkl}(j', l')$. Due to these considerations, in the matrices L^{ijkl}, LR^{ijkl}, G_{A}^{ijkl}, and G_{AB}^{ijkl}, we skip each complete row or column whose index is no candidate. Consequently, after computing a mapping from candidate sequence positions to matrix positions — independently for each sequence and for all significant base pairs, the sparsified algorithm operates on "contracted" matrices that contain only the candidate rows and columns. The first threshold θ_1 reduces the number of base pairs to a constant number of base pairs per sequence position; in total, quadratically many base pairs pass the filter. The thresholds on joint probabilities guarantee that each sequence position is candidate of only constantly many base pairs. Note that it is essential that we filter on joint probabilities and not on conditional probabilities as otherwise we could not infer a limit on the number of base pairs each sequence position is considered in (cf. Figure 3.2 in Chapter 3). By doing so, each position is considered only a constant number of times during the entire computation; this directly results in quadratic time complexity.

Theorem 4.2 *There are $O(n^2)$ entries $L^{ijkl}(j', l')$, $G_{\mathrm{A}}^{ijkl}(j', l')$, $G_{\mathrm{AB}}^{ijkl}(j', l')$, and $LR^{ijkl}(j', l')$ such that j' is a candidate of (i, j) and l' is a candidate of (k, l).*

Proof. Due to Lemma 3.2 (cf. Chapter 3) there are $O(n)$ many combinations i, j, j'. Analogously there are $O(n)$ combinations k, l, l' and therefore $O(n^2)$ combinations i, j, k, l, j', l' satisfying the conditions. □

Corollary 4.3 *The time and space complexity of computing all entries $L^{ijkl}(j', l')$, $G_{\mathrm{A}}^{ijkl}(j', l')$, $G_{\mathrm{AB}}^{ijkl}(j', l')$, and $LR^{ijkl}(j', l')$, D and F is $O(n^2)$.*

Proof. Clearly, the number of entries for all matrices except D is quadratically bounded due to Theorem 4.2. Matrix D has $O(n^2)$ entries as there are only a linear number of base pairs

$$D(ij,kl) = \max \begin{cases} -\infty \\ \text{if } \Pr[(i,j)|A] \geq \theta_1 \text{ and } \Pr[(k,l)|B] \geq \theta_1 \\ \quad \max\{L^{ijkl}(n_A-1, n_B-1), H^{ijkl}_{(j,l)}(n_A-1, n_B-1)\} + \tau(i,j,k,l) \end{cases}$$

$$L^{ijkl}(\bar{x},\bar{y}) = \max \begin{cases} -\infty \\ \text{if } \begin{array}{l} A_{j'} = B_{l'}, \text{wo-gap}^{ijkl}((\bar{x},\bar{y}),(i',k')) \text{ and} \\ \Pr[j' \in \text{loop}(i,j)|A] \geq \theta_2, \Pr[l' \in \text{loop}(k,l)|B] \geq \theta_2 \end{array} \\ \quad L^{ijkl}(\bar{x}-1, \bar{y}-1) + \sigma(j',l') \\ \text{for all } (i',j') \in P^A, (k',l') \in P^B \\ \quad \begin{array}{l} \Pr[(i',j')|A] \geq \theta_1, \Pr[(i',j') \in \text{loop}(i,j)|A] \geq \theta_3 \text{ and} \\ \text{with } \Pr[(k',l')|B] \geq \theta_1, \Pr[(k',l') \in \text{loop}(k,l)|B] \geq \theta_3 \text{ and} \\ (\bar{p},\bar{q}) = \text{mat-pos-bef}^{ijkl}(i',k') \text{ and } \text{wo-gap}^{ijkl}((\bar{p},\bar{q}),(i',k')) \end{array} \\ \quad L^{ijkl}(\bar{p},\bar{q}) + D(i'j',k'l') \end{cases}$$

$$G^{ijkl}_{\text{A}}(\bar{x},\bar{y}) = \max\{L^{ijkl}(\bar{x}-1,\bar{y}), G^{ijkl}_{\text{A}}(\bar{x}-1,\bar{y})\}$$
$$G^{ijkl}_{\text{AB}}(\bar{x},\bar{y}) = \max\{L^{ijkl}(\bar{x},\bar{y}-1), G^{ijkl}_{\text{A}}(\bar{x},\bar{y}-1), G^{ijkl}_{\text{AB}}(\bar{x},\bar{y}-1)\}$$
$$H^{ijkl}_{(i',k')}(\bar{x},\bar{y}) = \max \begin{cases} LR^{ijkl}(\bar{x},\bar{y}) & \text{if } \text{wo-gap}^{ijkl}((\bar{x},\bar{y}),(i',k')) \\ \max\{G^{ijkl}_{\text{A}}(\bar{x},\bar{y}), G^{ijkl}_{\text{AB}}(\bar{x},\bar{y})\} \end{cases}$$

$$LR^{ijkl}(\bar{x},\bar{y}) = \max \begin{cases} -\infty \\ \text{if } A_{j'} = B_{l'}, \Pr[j' \in \text{loop}(i,j)|A] \geq \theta_2, \Pr[l' \in \text{loop}(k,l)|B] \geq \theta_2 \\ \quad H^{ijkl}_{(j',l')}(\bar{x}-1,\bar{y}-1) + \sigma(j',l') \\ \quad L^{ijkl}(\bar{x}-1,\bar{y}-1) + \sigma(j',l') \quad \text{if } !\,\text{wo-gap}^{ijkl}((\bar{x}-1,\bar{y}-1),(j',l')) \\ \text{for all } (i',j') \in P^A, (k',l') \in P^B \\ \quad \begin{array}{l} \Pr[(i',j')|A] \geq \theta_1, \Pr[(i',j') \in \text{loop}(i,j)|A] \geq \theta_3 \text{ and} \\ \text{with } \Pr[(k',l')|B] \geq \theta_1, \Pr[(k',l') \in \text{loop}(k,l)|B] \geq \theta_3 \text{ and} \\ (\bar{p},\bar{q}) = \text{mat-pos-bef}^{ijkl}(i',k') \end{array} \\ \quad H^{ijkl}_{(i',k')}(\bar{p},\bar{q}) + D(i'j',k'l') \\ \quad L^{ijkl}(\bar{p},\bar{q}) + D(i'j',k'l') \quad \text{if } !\,\text{wo-gap}^{ijkl}((\bar{p},\bar{q}),(i',k')) \end{cases}$$

$$j' = \text{pos}^A_{(i,j)}(\bar{x}),\ l' = \text{pos}^B_{(k,l)}(\bar{y})$$
$$n_A = \left|\text{pos}^A_{(i,j)}\right|,\ n_B = \left|\text{pos}^B_{(k,l)}\right|$$

Figure 4.7: **Recursion equations on sparsified matrices.** Recursions for computing the significant strict EPMs and relaxed EPMs, respectively, on sparsified matrices; for additional definitions see Equation 3.4.

for each sequence (see Proposition 3.1). For the same reason, each matrix entry is computed in constant time: whenever we iterate over base pairs $(i, j) \in P^A$ and $(k, l) \in P^B$, we require $\Pr[(i, j)|A] \geq \theta_1$ and $\Pr[(k, l)|B] \geq \theta_1$. Hence, for fixed j and l, these iterations are constantly bounded. □

4.2 Chaining – Selecting a compatible subset of pattern matchings

Chaining selects a non-crossing and non-overlapping subset of EPMs. Our algorithm generalizes the chaining of ExpaRNA [HWBB09]. The chaining algorithm recursively fills the holes of all EPMs with other EPMs. For this purpose, it fills one $O(n^2)$ matrix for each hole and takes $O(Hn^2)$ time, where H is total number of holes with $H \ll n^2$. In contrast to ExpaRNA, there may exist more than one EPM ending at each sequence position pair, i.e. there is no one-to-one correspondence between EPMs and EPM's end positions. This is why each matrix requires additional steps in the order of the number of input EPMs E in ExpaRNA-P's chaining; the complexity of the generalized chaining algorithm is $O(H \cdot (n^2 + E))$. Since in the most general case, when we enumerate all suboptimal EPMs up to a maximal difference to the optimal score, $E \in O(n^2)$ is not guaranteed, we implement in addition several ways to control the number of EPMs. For example, ExpaRNA-P allows setting an *ad hoc* limit on this number. Furthermore, we suggest a heuristic strategy: for each sequence position pair, keep only the best EPM ending there. Consequently, typical use cases of ExpaRNA-P maintain the chaining complexity of ExpaRNA, i.e. $O(Hn^2)$.

4.3 Additional constraints on ExpaRNA-P's sparsified matrices

We further extend the algorithm ExpaRNA-P by including the `max-diff` constraint from LocARNA that was introduced in its current generality with the tool REAPR [WYB13]. In general, this constraint allows to specify regions in a dynamic programming matrix that will be considered during computation. The tool REAPR conducts whole genome realignments based on RNA sequence and structure by allowing a certain deviation (modeled by `max-diff`) from a given reference alignment.

Without a reference alignment, the `max-diff` constraint defines a band around the diagonal of the dynamic programming matrix. Thereby, it restricts the number of matrix cells that need to be filled and thus heuristically speeds up the computation. When utilizing EPMs as anchor constraints for computing a global alignment, it might be beneficial to use the `max-diff` constraint, as an EPM that matches the beginning of one sequence with the end of the other sequence is unlikely to be a good candidate as anchor constraint and thus should not be included in the set of traced EPMs.

The `max-diff` constraint is implemented by a maximal allowed deviation δ that defines an interval for each row in the matrix. In the unsparsified matrix, these intervals are defined by min-col(j') and max-col(j') for each row j' and only entries in this interval are filled. One possibility of initializing the interval for row j' is given by

$$\begin{aligned}
&\left|\frac{j'}{n} - \frac{l'}{m}\right| \leq \frac{\delta}{(n+m)/2} \quad \text{such that} \\
&\text{min-col}(j') = \frac{j'm}{n} - \frac{\delta m}{(m+n)/2},\ \text{max-col}(j') = \frac{j'm}{n} + \frac{\delta m}{(m+n)/2} \\
&\qquad \delta : \text{maximal allowed deviation defined by } \texttt{max-diff}
\end{aligned} \tag{4.3}$$

If the sequence lengths are the same, i.e. $m = n$ holds, we obtain min-col(j') $= j' - \delta$ and max-col(j') $= j' + \delta$. This defines a band of width 2δ around the diagonal of the DP matrix. For unequal sequence lengths, the width of the band is scaled accordingly. The minimal δ is chosen that guarantees that the intervals in two adjacent rows are connected, if this is not already fulfilled by Equation 4.3. This ensures that min-col(j') and max-col(j') are both monotonous increasing and the intervals are connected and non-empty.

The data structures for defining and traversing the sparsified matrices are constructed in a preprocessing step for each sequence separately. The `max-diff` constraint, however, defines regions in the sparsified matrices for each *base pair match* and thus we do not profit from preprocessing data structures in this case.

In the sparsified matrix, a position is *valid* in base pair match ($ij \sim kl$) if both corresponding sequence positions are candidates of the respective enclosing base pair of the base pair match (cf. Definition 3.2) and fulfill the `max-diff` constraint, i.e. the corresponding sequence positions are within the intervals specified by min-col and max-col. As only valid positions need to be considered, we can further limit the number of matrix positions. For that, we map a given interval to the sparsified matrix to identify the valid entries in the interval $\left[\text{col-idx-begin}^{ijkl}(\bar{x}), \text{col-idx-end}^{ijkl}(\bar{x})\right[$ for row $\bar{x}$:

- col-idx-begin$^{ijkl}(\bar{x})$ returns the matrix index $\bar{y}_b$ that stores the first candidate position that is greater than or equal to min-col(j') in sequence B, where $j' = \mathsf{pos}^A_{(i,j)}(\bar{x})$, i.e. $\bar{y}_b = \arg\min_{\bar{y}} \mathsf{pos}^B_{(k,l)}(\bar{y}) \geq \text{min-col}(j')$.
- col-idx-end$^{ijkl}(\bar{x})$ returns the matrix index $\bar{y}_e$ that stores the first candidate position that is greater than max-col(j') in sequence B, where $j' = \mathsf{pos}^A_{(i,j)}(\bar{x})$, i.e. $\bar{y}_e = \arg\min_{\bar{y}} \mathsf{pos}^B_{(k,l)}(\bar{y}) > \text{max-col}(j')$.

So, col-idx-begin$^{ijkl}(\bar{x})$ computes the first valid matrix index and col-idx-end$^{ijkl}(\bar{x})$ the index after the last valid matrix index in row $\bar{x}$. The column indices for the sparsified matrices can be computed in constant time by utilizing data structure $\mathsf{mat\text{-}idx\text{-}bef}^B_{(k,l)}$ (cf. Algorithm 4.1). Note that sequence position l'_{min} (l'_{max}) – the minimum (maximum) column for position j' –

is not necessarily in the range of the base pair (k, l) such that we need incorporate the cases that it is located before the left end k (line 3) or after the right end l of the base pair (line 6, $\min(l'_{min}, l)$ and $\min(l'_{max} + 1, l)$).

Algorithm 4.1 Identifying the valid interval for row $\bar{x}$ using the `max-diff` constraint.

Function: col-idx-begin$^{ijkl}(\bar{x})$

1: $j' \leftarrow \mathsf{pos}^{A}_{(i,j)}(\bar{x})$
2: $l'_{min} \leftarrow$ min-col(j')
3: **if** $l'_{min} \leq k$ **then**
4: $\quad \bar{y}_b \leftarrow 0$
5: **else**
6: $\quad \bar{y}_b \leftarrow \mathsf{mat\text{-}idx\text{-}bef}^{B}_{(k,l)}(\min(l'_{min}, l)) + 1$
7: **end if**
8: **return** $\bar{y}_b$

Function: col-idx-end$^{ijkl}(\bar{x})$

1: $j' \leftarrow \mathsf{pos}^{A}_{(i,j)}(\bar{x})$
2: $l'_{max} \leftarrow$ max-col(j')
3: **if** $l'_{max} < k$ **then**
4: $\quad \bar{y}_e \leftarrow 0$
5: **else**
6: $\quad \bar{y}_e \leftarrow \mathsf{mat\text{-}idx\text{-}bef}^{B}_{(k,l)}(\min(l'_{max} + 1, l)) + 1$
7: **end if**
8: **return** $\bar{y}_e$

Algorithm 4.2 Locating the first valid matrix position before sequence positions i' and k' including the `max-diff` constraint.

1: $(\bar{p}, \bar{q}) \leftarrow$ mat-pos-bef$^{ijkl}(i', k')$ # matrix position $(\bar{p}, \bar{q})$ without `max-diff`
2: $\bar{r} \leftarrow \bar{p}$
3: **while** $\bar{r} \geq 0$ **do** # check each row
4: $\quad \bar{s}_b \leftarrow$ col-idx-begin$^{ijkl}(\bar{r})$
5: $\quad \bar{s}_e \leftarrow$ col-idx-end$^{ijkl}(\bar{r})$ # $[\bar{s}_b, \bar{s}_e[$ interval for current row $\bar{r}$
6: $\quad$ **if** $\bar{s}_b < \bar{s}_e$ **and** $\bar{s}_b \leq \bar{q}$ **then** # valid interval found?
7: $\quad\quad \bar{s} \leftarrow \min(\bar{s}_e - 1, \bar{q})$ # determine correct column
8: $\quad\quad$ **return** $(\bar{r}, \bar{s})$ # return valid matrix position
9: $\quad$ **end if**
10: $\quad$ --$\bar{r}$
11: **end while**

After having introduced an efficient method to map the valid intervals to the sparsified matrix, we describe how to efficiently locate the first valid matrix position before sequence positions i' and k' when the `max-diff` constraint is used to further restrict the number of cells in the dynamic programming matrix (cf. Algorithm 4.2). We cannot guarantee that the matrix position $(\bar{p}, \bar{q}) =$ mat-pos-bef$^{ijkl}(i', k')$ that lies before sequence positions i' and k' (cf. Equation 3.4) is valid, as it might not fulfill the `max-diff` constraint. Thus, we start at $(\bar{p}, \bar{q})$ and check row-wise whether there exists a valid position whose second index is located before or at $\bar{q}$. By utilizing the mapped intervals, each row can be checked in constant time (line 6). An illustrative example is shown in Figure 4.8. The blue parts show the candidate positions and the light red parts the intervals that fulfill the `max-diff` constraint. The overlapping and thus valid parts are highlighted in red. After considering the base pair match $1a$), we identify the matrix position $(\bar{p}, \bar{q})$ that lies before the left ends i' and k' of the base pairs. As the interval identified with Algorithm 4.1 is empty, no valid positions can be found in row $\bar{p}$. The algorithm

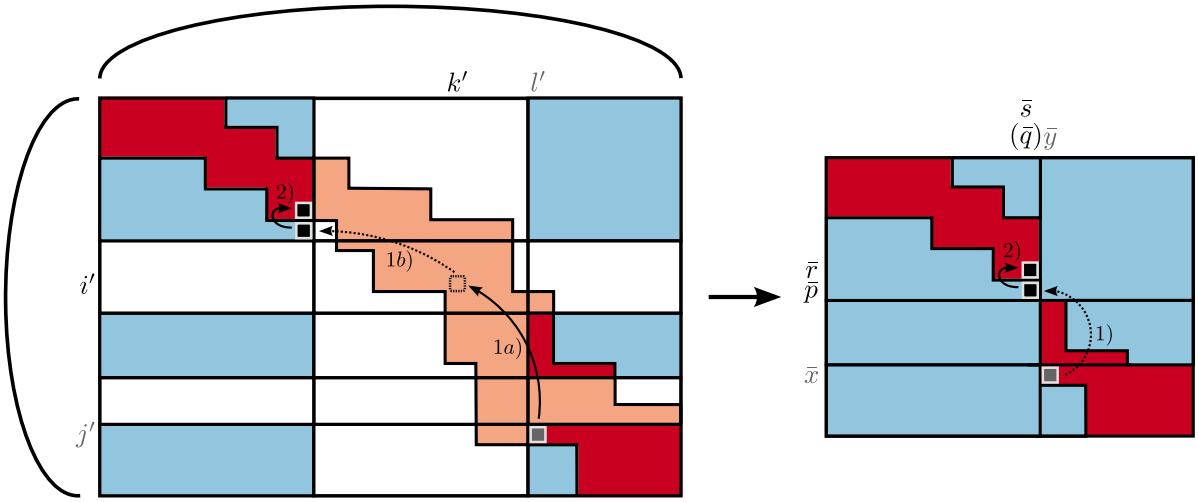


Figure 4.8: **Illustration of the `max-diff` constraint in the sparsified matrix.** The blue parts show the candidate positions and the light red parts the intervals that fulfill the `max-diff` constraint. The overlapping and thus valid parts are highlighted in red. First we compute the matrix position $(\bar{p}, \bar{q})$ that lies before i' and k', the left ends of the base pair match $1a)$. As this matrix position does not fulfill the `max-diff` constraint, we check row-wise until the valid entry $(\bar{r}, \bar{s})$ is found (cf. Algorithm 4.2).

moves to the next line $\bar{r}$ where a non-empty interval is found and the matrix position $(\bar{r}, \bar{s})$ is returned.

4.4 Results

We implemented ExpaRNA-P and the chaining algorithm in C++. In particular, we implemented two versions of the traceback: the suboptimal traceback and a heuristic version that, for each match $(i \sim k)$, considers only the optimal EPM ending at that match. Our tool supports two ways to control the EPM enumeration by the suboptimal traceback: either by defining the maximum score difference to the optimal score or the maximum number of EPMs.

In order to assess the performance of ExpaRNA-P, we designed the following pipeline: In a first step we compute the significant EPMs with ExpaRNA-P and use the chaining algorithm to extract from these EPMs an optimal non-overlapping and non-crossing subset. Then we compute a sequence structure alignment that includes all matches of the chained EPMs. For this purpose, we utilize the EPMs as anchor constraints for LocARNA [WRH+07]. Consequently, LocARNA runs much faster, since each anchor reduces the alignment space. We call our pipeline ExpLoc-P in correspondence with the analogous approach ExpLoc [HWBB09], which utilizes ExpaRNA anchors.

We performed all benchmarks over the pairwise alignment instances of the BRAliBase 2.1 bench-

Table 4.1: **Comparison of ExpLoc-P variants.** Total runtimes of different ExpLoc-P variants (see text) and average SPS scores over the BRAliBase 2.1 benchmark set. The runtimes are split into (a) time for preprocessing, (b) time for computing and chaining the EPMs and (c) time for the LocARNA alignments with anchor constraints. Table adapted from [OMH+14].

ExpLoc-P variant	1	2	3	4	5
total SPS	0.86	0.84	0.86	0.84	0.84
total time	3.5h	3.0h	3.7h	3.1h	3.1h
(a) preprocessing	0.6h	0.6h	0.6h	0.6h	0.6h
(b) EPM calculation	0.4h	0.5h	0.4h	0.5h	0.5h
(c) LocARNA alignment	2.5h	1.9h	2.7h	2.0h	2.0h

mark set [WMS06, GWW05]. To measure the quality of the calculated alignment in comparison with the (for each instance) known reference alignment, BRAliBase 2.1 provides the scoring tool compalignp. It computes the similarity between the two alignments as sum-of-pairs score (SPS). Identical alignments receive the SPS score 1; alignments without any correspondence, 0. In this way, we evaluated different variants of our method and later compare them with existing tools. At the same time, we opposed quality to runtime.

4.4.1 Impact of EPM selection on the performance

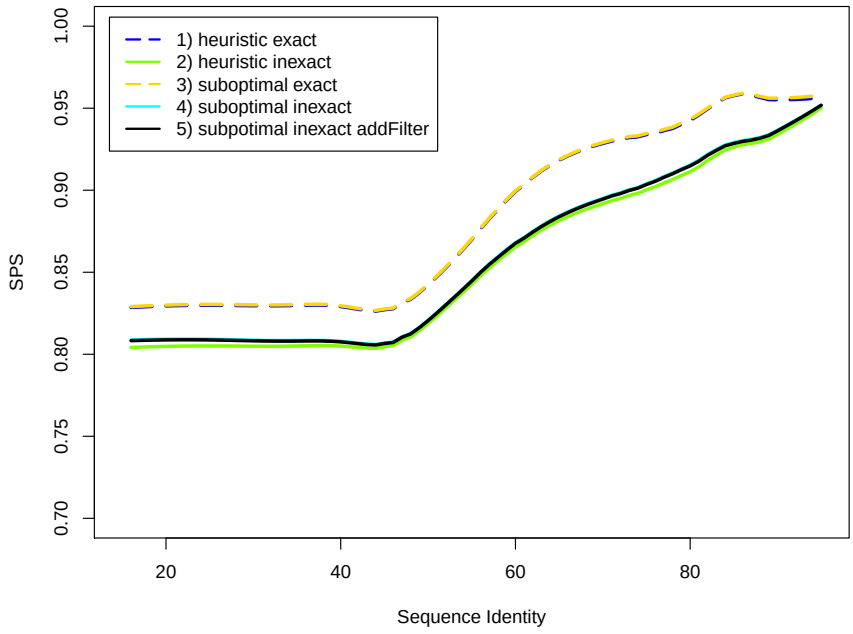
We study five ExpLoc-P variants, where we generate anchor constraints respectively by

1) heuristic traceback with exact matches
2) heuristic traceback with inexact structure matches
3) suboptimal traceback with exact matches
4) suboptimal traceback with inexact structure matches
5) suboptimal traceback with inexact structure matches and the additional second filter step

In particular, we compare exact modes (1,3), which follow the strict EPM definition, and inexact modes (2,4,5), which allow mismatches at structure positions (relaxed EPMs).

The parameters were selected ad-hoc without parameter learning. In particular, we set the cutoff probabilities to $\theta_1 = \theta_2 = \theta_3 = 0.01$ to predict less false positives. Furthermore, we enumerated EPMs that have a score of at least 90 and fix the maximal number of traced EPMs in the suboptimal traceback to 100. The scoring – as defined in Equation 4.1 and 4.2 – was instantiated by setting the structure mismatch score str-mm to -10 for structure mismatches in inexact modes. Furthermore we set $\alpha_1 = 1$, $\alpha_2 = 5$ and $\alpha_3 = 5$ in order to favor structured regions.

(a)



(b)

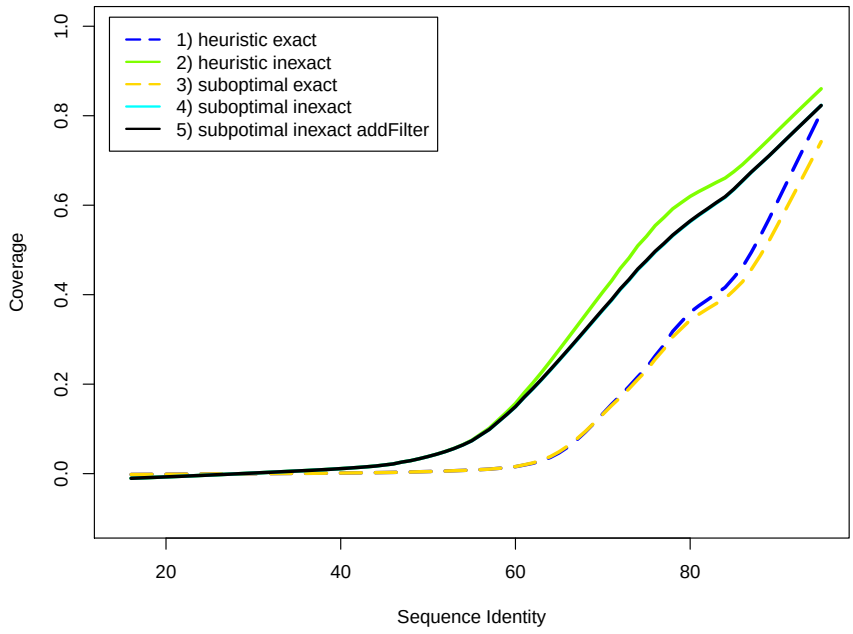
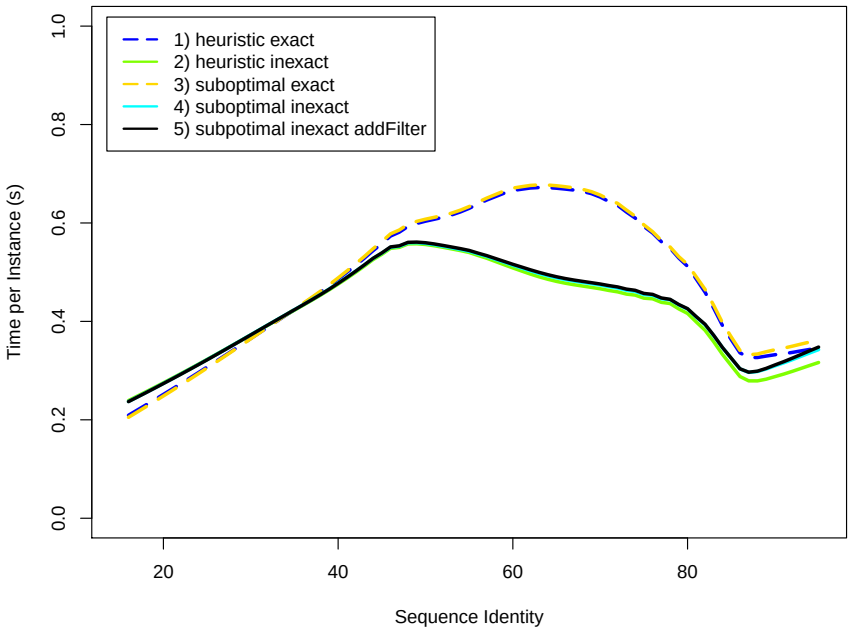


Figure 4.9: **Comparison of ExpLoc-P variants.** (a) Alignment quality (SPS) vs. sequence identity. (b) Coverage vs. sequence identity. Dependencies are visualized as lowess curves. Figure taken from [OMH+14].

(a)



(b)

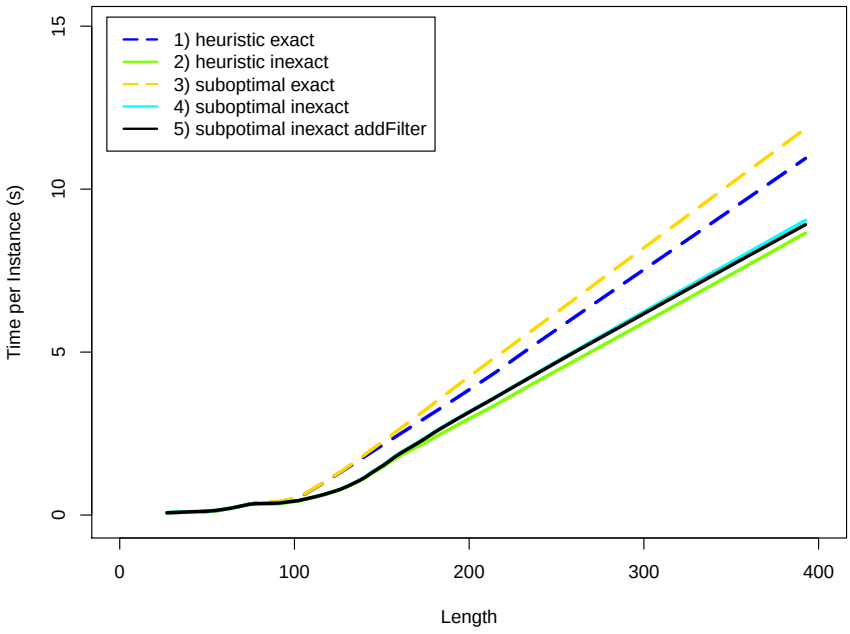


Figure 4.10: **Comparison of ExpLoc-P variants.** (a) Runtime vs. sequence identity (b) Runtime vs. length. The length of a benchmark instance is defined as the average length of its two sequences. Dependencies are visualized as lowess curves. Figure taken from [OMH+14].

In addition to SPS and runtime, we computed the coverage for each benchmark instance – consisting of sequences A and B. For this purpose, we define *coverage* as the fraction of nucleotides that are matched by the best chain of EPMs $\mathcal{C} = \bigcup(\mathcal{M}, \mathcal{S})$:

$$\text{coverage} = \frac{\sum_{(\mathcal{M},\mathcal{S})\in\mathcal{C}} |\mathcal{M}|}{\min(A, B)} \tag{4.4}$$

Figure 4.9a shows the alignment quality (SPS) versus the sequence identity; we visualized the dependency by estimating a lowess curve [Cle81] for each series of benchmark evaluations. Overall, we observed that the difference between the suboptimal and heuristic traceback is not significant, solely for inexact modes, the suboptimal traceback leads to slightly better results. Furthermore, in inexact modes the additional second filter step did not change the quality significantly. Exact modes produced better alignments, however these modes generated much less anchor constraints for low sequence identity regions; in turn, the speedup decreases in these modes. This effect is visible in Figure 4.9b, which plots the estimated coverage vs. the sequence identity. The exact modes predict EPMs only for sequence identity values above 60%. For the inexact modes, we obtained much higher coverage; notably, we predicted many more relaxed EPMs than strict EPMs for the sequence identity interval from 40-60%.

In Table 4.1, we report total runtimes and average SPS scores of different ExpLoc-P variants over the entire benchmark set. Furthermore, we provide single timings for (a) preprocessing, (b) computing and chaining the EPMs and (c) subsequent LocARNA alignments.

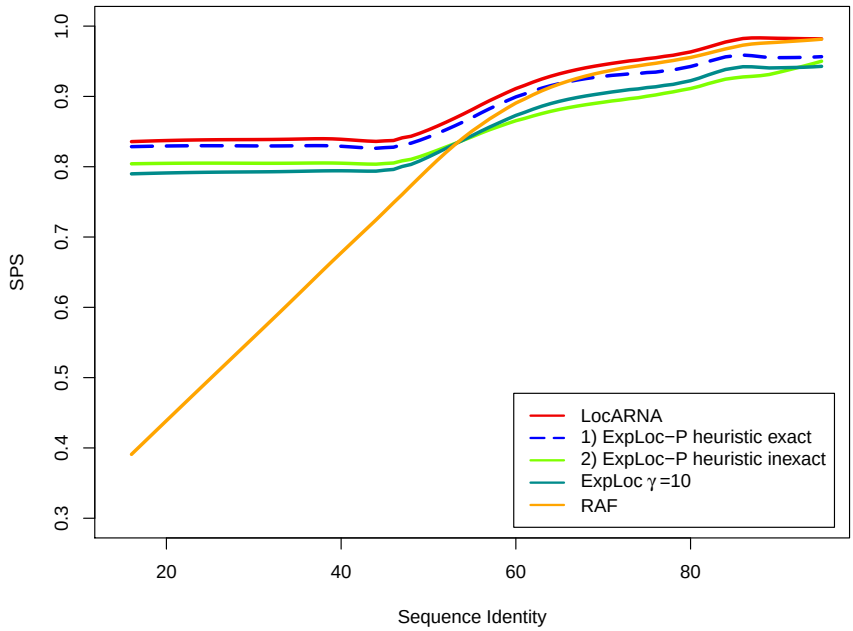
The differences in coverage directly impact the runtimes of the different variants, but not as pronounced, since – like one would expect for many real world applications – the benchmark set contains many high identity sequences. Consequently, relaxed EPMs significantly reduced the runtime for instances with sequence identity between 50-80% (Figure 4.10a). Furthermore, the heuristic traceback was slightly faster than the suboptimal one for long RNA sequences (Figure 4.10b), while suboptimal traceback could not significantly improve the alignment quality in this setting. Consequently, for this specific benchmark, the two variants with heuristic traceback turned out to provide the best balance of quality and speed up.

4.4.2 Comparison with other tools

We benchmarked three existing approaches: LocARNA [WRH$^+$07], ExpLoc [HWBB09], and RAF [DFB08]. LocARNA without anchors serves as base line approach; in contrast to ExpLoc-P, ExpLoc identifies EPMs in a single predicted structure for each RNA (using ExpaRNA); and RAF is currently the fastest Sankoff-style alignment approach due to its heuristic filtering based on sequence alignments. We compared these approaches with ExpLoc-P variants 1 and 2, which performed best in the previous section.

Table 4.2 summarizes the results; we report the speedup over LocARNA, total runtime, and

(a)



(b)

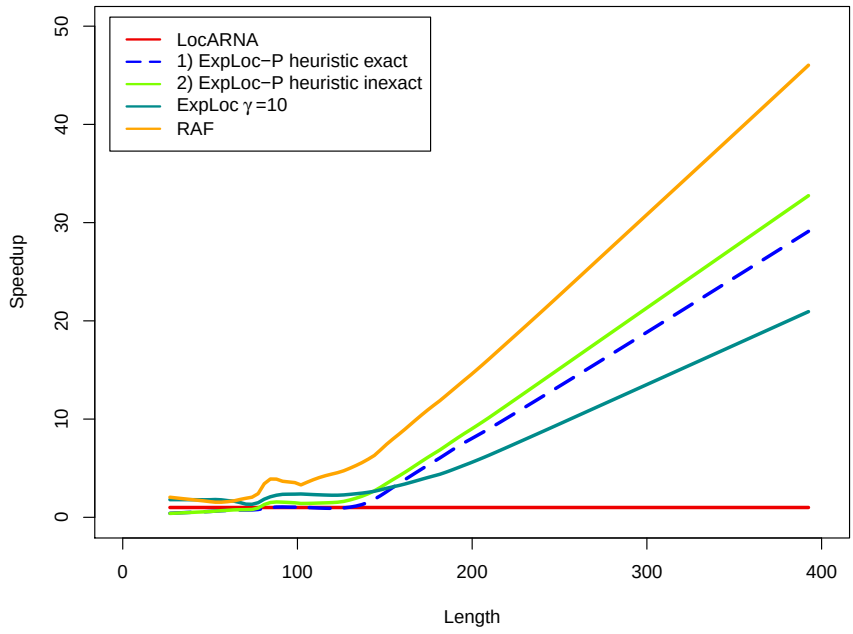


Figure 4.11: **Comparison with sequence-structure alignment methods.** (a) Comparison of alignment quality vs. sequence identity. (b) Comparison of speed up over LocARNA vs. length. Dependencies are visualized as lowess curves. Figure taken from [OMH+14].

Table 4.2: **Comparison of RNA alignment methods.** We report speedup over LocARNA, total runtime, and average alignment quality (SPS) over the BRAliBase 2.1 benchmark set. Table taken from [OMH+14].

	LocARNA	ExpLoc-P (variant 1)	ExpLoc-P (variant 2)	ExpLoc $\gamma = 10$	RAF
speedup	1	3.9	4.6	5.0	14.4
runtime	13.8h	3.5h	3.0h	2.8h	1.0h
SPS	0.87	0.86	0.84	0.81	0.86

average alignment quality (SPS) across the entire benchmark set (Opteron 2356, 2.3 GHz, single thread). Figure 4.11a shows the behavior of the SPS dependent on the sequence identity. LocARNA aligned with the best quality at the expense of the highest computation time. The best alignment quality that has been obtained with ExpLoc in [HWBB09] has been achieved with parameter setting $\gamma = 10$ (the minimal EPM size). Even this quality is significantly lower than the one for the two variants of ExpLoc-P (0.81 vs. 0.84 and 0.86). Moreover, the overall speedup for this setting is not much higher than the speedups for ExpLoc-P. Although RAF achieved the best speedup of 14.4, the quality drops tremendously for sequence similarities below 50%. The quality drop of RAF alignments at low sequence identities is strongly reminiscent of pure sequence alignment methods. Thus, we conjecture that the specific use of sequence-based heuristics by RAF, while guaranteeing sequence alignment like run-time behavior, compromises RAF's use for 'hard' RNA alignment instances that require structure-based alignment methods.

Furthermore, we investigated the dependency of the lengths of the input sequences on the speedup (see Figure 4.11b). As expected, the speedup increased for longer input sequences. For RNA sequences longer than 150 bases, we obtained a significantly better speedup with both variants of ExpLoc-P compared with ExpLoc. For the longest input sequences, ExpLoc-P achieved respective speedups of 32 and 35 for the exact and inexact mode, and RAF of almost 50.

Furthermore, Figure 4.12 compares the speedup distributions of ExpLoc-P variants 1 and 2 as boxplot. The benchmark instances are classified according to their sequence length in the intervals (0,50], (50,100], (100,150], (150,200], (200,250], (250,300], (300,350], and (350,400]. For sequences of lengths greater than 150 we achieve substantial speedup. The inexact mode is also superior to the exact mode for these input sequences.

To summarize, ExpLoc-P provided the best trade-off between alignment quality and speedup in this setting; robustly, it maintained high alignment quality over the entire range of sequence identities; finally, it proofed to be particularly suited for long instances.

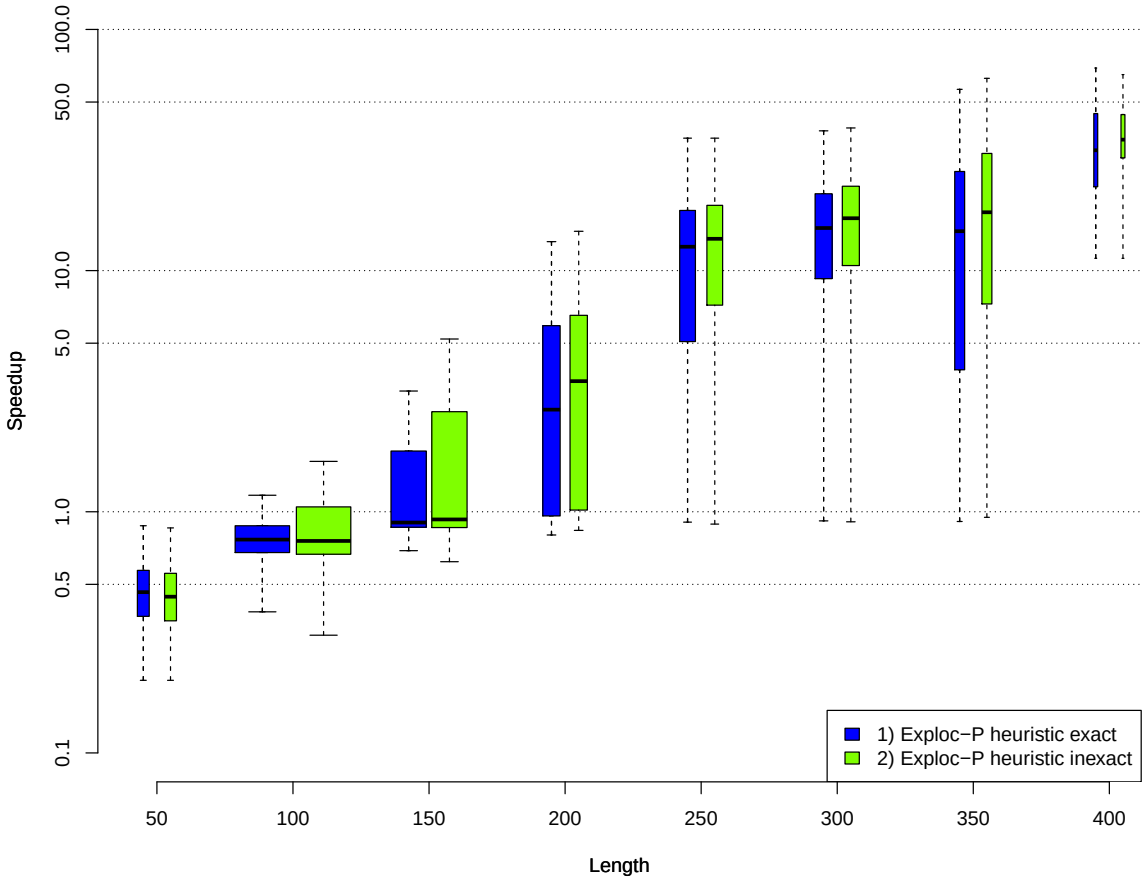


Figure 4.12: **Speedup potential increases with the sequence lengths.** Boxplot for the speedup over LocARNA vs. sequence lengths for ExpLoc-P variants. The speedup is measured relative to the speed of LocARNA. The width of the boxes are scaled according to the number of benchmark instances in each interval. For sequences longer than 150, a considerable speedup is obtained for both variants. Furthermore, the inexact variant shows a larger potential for speedup for these input sequences. Figure taken from [OMH+14].

4.5 Discussion

We have introduced the algorithm ExpaRNA-P that very efficiently identifies exact pattern matches in RNAs by matching and folding them simultaneously. The method is a major achievement over previous approaches (including the "predecessor" ExpaRNA) that – without being more efficient – are much less flexible, since they require a priori known or unreliably predicted structure. Due to its novel ensemble-based sparsification, the algorithm ExpaRNA-P has only a very low (quadratic) time and space complexity, equaling sequence alignment.

We have developed two major variants of this method; one requires strict matches in all

positions of an EPM (strict EPMs), the other relaxes this (therefore, relaxed EPMs) to allow mismatches at structural positions. The latter supports compensatory mutations, which are highly relevant in RNA structure analysis in general.

Our benchmarks study EPMs as anchor constraints to speed up RNA structure alignments (in the form of simultaneous alignment and folding by LocARNA). EPMs from structure ensembles have turned out to be substantially more reliable than EPMs from fixed structures. At comparable speed ups, this results in increased quality. Most importantly, the novel approach keeps up the alignment quality even for sequences of low identity, which is ultimately decisive for structure alignment. In striking contrast, the alignment quality of the similarly fast alignment tool RAF breaks down – very much like pure sequence alignment.

We have implemented rigorous suboptimal traceback, which provides extensive control of the set of enumerated EPMs. For example, this level of control is required in the analysis of structural variants common to the RNAs. In addition, we have developed a heuristic traceback that restricts the number of enumerated EPMs. This variant performs almost indistinguishable in our benchmark. Being much faster than the rigorous method, it offers the best speed-quality balance in such settings. Furthermore, we demonstrated how to integrate further constraints on ExpaRNA-P's dynamic programming matrices in addition to the ensemble-based sparsification.

Finally, reliable EPM-based anchor constraints as computed by ExpaRNA-P can be combined advantageously with other RNA alignment tools such as RAF. While for LocARNA the constraints yield a considerable speedup, the combination with RAF has the potential to improve RAF's poor alignment quality for low sequence similarity.

CHAPTER 5

Fast and accurate simultaneous alignment and folding

Ever since Sankoff's algorithm [San85] for simultaneous alignment and folding was proposed in 1985, a lot of effort was put into reducing its extreme time complexity. A detailed description of the different approaches is given in Section 2.4. Since separating structure and alignment computation introduces the risk of predicting an incorrect alignment, the favored approach computes the structure and alignment simultaneously. Developing fast algorithms for simultaneous alignment and folding that maintain good alignment quality still poses a big challenge especially for RNA families with low sequence identity. A lightweight energy model [HBS04] that scores structural features based on base pair probabilities constitutes a first step towards efficient approaches that keep up good alignment quality. However, none of the existing algorithms that apply a lightweight energy model implement the full flexibility of Sankoff's model. In this chapter, we introduce two novel algorithms for computing sequence-structure alignments simultaneously. PARSE enables Sankoff's flexible structure prediction while applying a lightweight energy model. The sparsified variant SPARSE utilizes in addition the efficient ensemble-based sparsification (cf. Section 3) to significantly speed up the computation.

All subsequent algorithms are described for sequences A and B with respective lengths n and m ($n \geq m$) and associated base pair sets P^A and P^B, and structures R and T. Formal definitions of sequence alignment (denoted by $\mathcal{A}$) and sequence-structure alignment (denoted by $(\mathcal{A}, R, T)$) are provided in Section 2.3. The description in this chapter is based on [WOM+15]. A preliminary version was published in [WSM+13].

Contributions We present the novel algorithm PARSE, which is the first approach for fast simultaneous alignment and folding that combines the original flexible structure prediction of Sankoff's algorithm with the lightweight energy model of PMcomp [HBS04] to efficiently score

structure contributions. To initially improve the time complexity from $O(n^6)$ to $O(n^4)$, we additionally integrate a base pair filter that was proposed in LocARNA [WRH$^+$07]. Crucially, this filter does not impair the quality of the sequence-structure alignment. Additionally, we introduce the sparsified variant SPARSE that incorporates the novel ensemble-based sparsification introduced in Section 3. The in-loop probabilities enable to identify efficiently in a preprocessing step those unpaired bases and base pairs that are likely to occur in a particular loop and thus allow to restrict the search space. This technique solely utilizes properties of the structure ensemble and does not rely on sequence-based heuristics that could cause mis-alignments for RNA families with low sequence conservation. These are introduced for instance in [Hol05, DE06, HSM07] and in the tool RAF [DFB08], which is with quadratic runtime currently the fastest tool for simultaneous alignment and folding. Integrating the novel ensemble-based sparsification in SPARSE leads to a drastic reduction to quadratic runtime without utilizing sequence-based heuristics. Remarkably, the complexity of SPARSE matches the time complexity of sequence alignment. To incorporate this novel sparsification technique into a sequence-structure alignment tool requires compatible individual structure prediction for the two input sequences as implemented in PARSE. Notably, directly integrating it into PMcomp or LocARNA is not possible.

Overview of results To evaluate the effectiveness of our novel algorithm SPARSE, we conduct extensive benchmarks on the well-established benchmark set BRAliBase 2.1 [WMS06, GWW05]. We show that SPARSE achieves an alignment quality comparable with LocARNA, but unlike RAF maintains it even for hard instances with low sequence conservation. Moreover, SPARSE achieves roughly the same speed up as RAF of around 4. The enhanced structure prediction flexibility of SPARSE over LocARNA is reflected in a significant improvement of the structure prediction accuracy.

5.1 Sankoff's algorithm and Sankoff-style alignment

The idea to simultaneously align and fold RNA sequences was introduced in Sankoff's algorithm [San85] that computes an alignment $\mathcal{A}$ and RNA structures R and T simultaneously for two given RNA sequences A and B. Even though this approach has a high $O(n^6)$ time and $O(n^4)$ space complexity, it is still considered the gold standard for RNA alignment. Its objective function combines the free energy of the predicted structures for the input sequences in a loop-based model [MSZT99] with a scoring of the edit distance of the predicted alignment. To get a biologically meaningful result and to simplify computation, dependencies between the alignment and structures are introduced. Each k-loop has exactly one k-loop counterpart to which it is aligned or the whole k-loop is deleted or inserted. Thus, it is not possible to align one k-loop to positions in more than one k-loop in the other sequence. Furthermore,

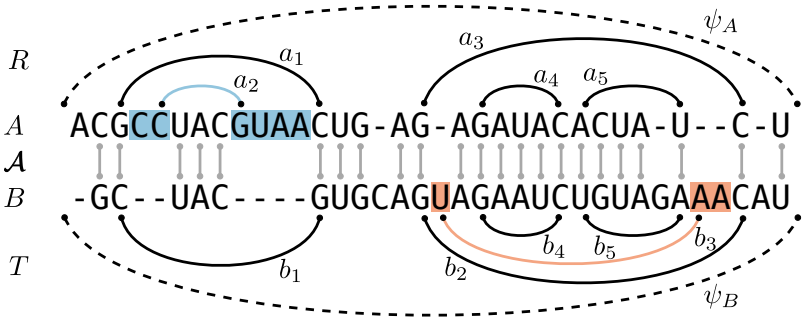


Figure 5.1: **Visualization of Sankoff's restrictions on alignment and structures.** Alignment of sequences A and B with associated structures R and T, where a_k and b_k ($1 \leq k \leq 5$) are base pairs. ψ_A and ψ_B denote the pseudo base pairs that cover the entire sequences. As required by Sankoff's algorithm, all interior base pairs of multiloops (a_4, a_5, b_4, b_5) and external base pairs (a_1, a_3, b_1 and b_2) are covered by $(\mathcal{A}, R, T)$. The positions highlighted in blue belong to the 2-loop closed by base pair a_1 and base pair a_2 can only be deleted together with the entire 2-loop (loop deletion). Analogously, the positions in red show the case of a loop insertion within a 2-loop. Figure adapted from [WOM+15].

the general shape of the RNA structures should be identical. This *branching configuration* is maintained if all external base pairs and the interior base pairs of multiloops of the two predicted RNA structures are covered by the sequence-structure alignment (cf. Section 2.3). However, if only the number of base pairs in the stem regions of two RNA structures differ, they still have the same overall shape. Thus, insertion and deletion of 2-loops is supported, but with the requirement above, only if the interior base pair is inserted or deleted together with all unpaired positions in the loop. This operation corresponds to a shortening or elongation of stems. Insertion and deletion of 2-loops are defined in Definition 5.1 and illustrated in Figure 5.1. The precise restrictions on the predicted alignment and structures in Sankoff's algorithm are given in Definition 5.2.

Definition 5.1 (Insertion and deletion of 2-loops)
Let $(\mathcal{A}, R, T)$ be a sequence-structure alignment, where $\mathcal{A}$ is a sequence alignment and R and T are structures for sequences A and B, respectively (cf. Definition 2.7). A sequence-structure alignment $(\mathcal{A}, R, T)$ of sequences A and B deletes [inserts] the entire 2-loop closed by base pair $(i, j) \in R$ [$(k, l) \in T$] iff the base pair closes a 2-loop and the base pair and all bases in $\text{loop}_A(i, j)$ [$\text{loop}_B(k, l)$] are deleted [inserted].

Definition 5.2 (Sankoff's restrictions on alignment and structures)
Let $(\mathcal{A}, R, T)$ be a sequence-structure alignment. $(\mathcal{A}, R, T)$ is a structure alignment triple iff it satisfies Sankoff's restrictions, i.e. iff

1. for all $(i, j) \in R$: $(\mathcal{A}, R, T)$ covers (i, j) or deletes the entire 2-loop closed by $\text{parent}_A(i, j)$
2. for all $(k, l) \in T$: $(\mathcal{A}, R, T)$ covers (k, l) or inserts the entire 2-loop closed by $\text{parent}_B(k, l)$

An extension to multiple alignment is also suggested in [San85] with an even more extreme $O(n^{3N})$ time and $O(n^{2N})$ space complexity, where N denotes the number of sequences.

PMcomp [HBS04] simplifies the original Sankoff algorithm in two ways. Instead of a loop-based energy model, PMcomp uses a lightweight energy model based on the base pair probability matrices – as computed by McCaskill's algorithm (cf. Section 2.2.3) – of the two input sequences. Thereby, the free energy contribution in the loop-based energy model is estimated by a product of the base pair probabilities while assuming their independence for the sake of computational simplicity. For that, the weight of base pair (i, j) is defined by $\Psi_{ij}^{A} = \log(\Pr[(i,j)|A]/p_0)$, where p_0 is the background probability for base pairs, i.e. the minimum base pair probability that is considered significant.

To define the score of a sequence-structure alignment, we first introduce the unstructured part $\mathcal{A}_{RT}^{u}$ of the alignment, where $(i,k) \in \mathcal{A}_{RT}^{u}$ iff $\forall j : (i,j) \notin R$ and $(j,i) \notin R$, and $\forall l : (k,l) \notin T$ and $(l,k) \notin T$. Since log odds scores are applied, the base pair weights of the predicted structures are summed up in addition to the regular sequence alignment score for the unstructured part $\mathcal{A}_{RT}^{u}$.[1]

$$\text{score}((\mathcal{A}, R, T)) = \sum_{(i,j)\in R} \Psi_{ij}^{A} + \sum_{(k,l)\in T} \Psi_{kl}^{B} + \sum_{(i,k)\in \mathcal{A}_{RT}^{u}} \sigma(i,k) + n_{\text{indel}}^{\mathcal{A}}\gamma \tag{5.1}$$

The second simplification of PMcomp reduces the flexibility of Sankoff's original structure prediction. Whereas Sankoff's algorithm predicts two individual but compatible structures, PMcomp outputs a single consensus structure for both input sequences. This is a strong restriction as this means that all predicted base pairs have to be covered by the sequence-structure alignment. In the more expressive Sankoff's algorithm, only interior base pairs of multiloops and external base pairs have to be covered such that insertions and deletions of entire 2-loops are allowed. These are particularly important as they correspond to the elongation or shortening of stems. PMcomp does not support loop deletions or insertions at all.

The original PMcomp algorithm shown in Figure 5.2a first computes the best score of all subsequence combinations $A_{i..j}$ and $B_{k..l}$. The first three cases of $\bar{M}$ are analogous to sequence alignment, whereas the last case considers matches of base pairs. Those are computed in matrix $\bar{D}$ by adding contributions Ψ_{ij}^{A} and Ψ_{kl}^{B} for the matched base pairs (i,j) and (k,l) and $\bar{M}(i+1, j+1, k-1, l-1)$ for the part enclosed by the base pair match. The original formulation of PMcomp considers $\bar{M}^{ijkl}$ for each combination of i,j,k and l, resulting in $O(n^6)$ time complexity if we do not constrain the base pair sets P^A and P^B because then each entry needs quadratic time. The space complexity is dominated by storing all $\bar{D}$ matrices since at each time only one $\bar{M}$ matrix needs to be stored, which results in $O(n^4)$ space complexity.

When applying sparsification, a variant of the original PMcomp algorithm is required (Fig-

[1] Note that we slightly modified PMcomp's original alignment and folding score to simplify the presentation.

(a)

$$\text{for all } i \text{ with } 1 < i < n,\ j \text{ with } i < j < n \text{ and}$$
$$k \text{ with } 1 < k < m,\ l \text{ with } k < l < m\text{:}$$
$$\bar{M}(i,j,k,l) = \max \begin{cases} \bar{M}(i,j-1,k,l-1) + \sigma(j,l) \\ \bar{M}(i,j,k,l-1) + \gamma \\ \bar{M}(i,j-1,k,l) + \gamma \\ \max\limits_{\substack{i' \geq i \\ k' \geq k}} \bar{M}(i,i'-1,k,k'-1) + \bar{D}(i'j,k'l) \end{cases}$$
$$\bar{D}(ij,kl) = \bar{M}(i+1,j+1,k-1,l-1) + \Psi^A_{ij} + \Psi^B_{kl}$$

(b)

$$\text{for all } (i,j) \in P^A,\ (k,l) \in P^B \text{ and}$$
$$j' \text{ with } i < j' < j,\ l' \text{ with } k < l' < l\text{:}$$
$$\hat{M}^{ijkl}(j',l') = \max \begin{cases} \hat{M}^{ijkl}(j'-1,l'-1) + \sigma(j',l') \\ \hat{M}^{ijkl}(j',l'-1) + \gamma \\ \hat{M}^{ijkl}(j'-1,l') + \gamma \\ \max\limits_{\substack{(i',j') \in P^A, \\ (k',l') \in P^B}} \hat{M}^{ijkl}(i'-1,k'-1) + \hat{D}(i'j',k'l') \end{cases}$$
$$\hat{D}(ij,kl) = \hat{M}^{ijkl}(j-1,l-1) + \Psi^A_{ij} + \Psi^B_{kl}$$

Figure 5.2: **PMcomp recursion equations.** (a) Original PMcomp algorithm and (b) slightly modified version suitable for sparsification. While $\bar{M}(i,j,k,l)$ is computed for all combinations of i, j, k and l, $\hat{M}^{ijkl}$ is only computed if base pairs (i,j) and (k,l) occur in the respective base pair set. Figure adapted from [WOM+15].

ure 5.2b). The main difference is that $\hat{M}^{ijkl}$ is only computed if base pairs (i,j) and (k,l) occur in the respective base pair set. When a fixed base pair filter is applied (cf. Section 3.1), a linear number of base pairs is retained in P^A and P^B such that only $O(n^2)$ many $\hat{M}$ matrices need to be filled. Furthermore, the time to compute one entry in the $\hat{M}$ matrix is reduced to constant time. Thus, this reduces the overall time complexity to $O(n^4)$. Keeping at any time only one $\hat{M}$ matrix and storing all $\hat{D}$ matrices requires $O(n^2)$ space.

For computing multiple alignments, a progressive strategy is chosen and implemented in the tool PMmulti. For that, a consensus base pair probability matrix is computed from the sequence-structure alignment in each step of the progressive phase by taking the geometric mean of the probabilities of all matched base pairs and using those transformed probabilities as a basis to compute the base pair weights.

$$\tilde{D}_S(ij,kl) = \max \begin{cases} \tilde{M}^{ijkl}(j-1,l-1) \\ \tilde{I}_{\mathrm{A}}^{ijkl}(j-1) \\ \tilde{I}_{\mathrm{B}}^{ijkl}(l-1) \end{cases}$$

$$\tilde{M}^{ijkl}(j',l') = \max \begin{cases} \tilde{M}^{ijkl}(j'-1,l'-1) + \sigma(j',l') \\ \tilde{M}^{ijkl}(j',l'-1) + \gamma \\ \tilde{M}^{ijkl}(j'-1,l') + \gamma \\ \max\limits_{\substack{(i',j')\in P^A, \\ (k',l')\in P^B}} \tilde{M}^{ijkl}(i'-1,k'-1) + \tilde{D}_S(i'j',k'l') + \Psi^A_{i'j'} + \Psi^B_{k'l'} \end{cases}$$

$$\tilde{I}_{\mathrm{A}}^{ijkl}(j') = \max \begin{cases} \tilde{I}_{\mathrm{A}}^{ijkl}(j'-1) + \gamma \\ \max\limits_{(i',j')\in P^A} (i'-i+1)\gamma + \tilde{D}_S(i'j',kl) + \Psi^A_{i'j'} \end{cases}$$

$$\tilde{I}_{\mathrm{B}}^{ijkl}(l') = \max \begin{cases} \tilde{I}_{\mathrm{B}}^{ijkl}(l'-1) + \gamma \\ \max\limits_{(k',l')\in P^B} (k'-k+1)\gamma + \tilde{D}_S(ij,k'l') + \Psi^B_{k'l'} \end{cases}$$

Figure 5.3: **Recursion equations.** PARSE recursions for computing the best structure alignment triple. These equations are visualized in Figure 5.4. Figure adapted from [WOM+15].

5.2 Optimizing Sankoff-style alignment

5.2.1 PARSE – Flexible lightweight simultaneous alignment and folding

To incorporate the full flexibility of Sankoff's model, we extend the PMcomp recursions (cf. Figure 5.3) with the loop deletion and insertion case. To cover these new cases, matrices $\tilde{I}_{\mathrm{A}}$ and $\tilde{I}_{\mathrm{B}}$ are introduced, see recursion equations in Figure 5.3 and a visualization in Figure 5.4. The entries are recursively defined for all $(i,j) \in P^A$, $(k,l) \in P^B$, j' $(i < j' < j)$, and l' $(k < l' < l)$. As an entire 2-loop is deleted in $\tilde{I}_{\mathrm{A}}$ we either delete the current position j' or jump over a base pair that is deleted including the whole part before it. The recursion for $\tilde{I}_{\mathrm{B}}$ is analogous. Since in this model not all base pairs are necessarily covered, we cannot include the score of a base pair match already in matrix $\tilde{D}_S$ as it was done in PMcomp. Thus, we change the definition of $\tilde{D}_S$ matrix entries to consider only the part between base pairs (i,j) and (k,l). The score contribution of a base pair match is considered in matrix $\tilde{M}$, whereas the score for inserting or deleting a base pair is added in matrix $\tilde{I}_{\mathrm{A}}$ and $\tilde{I}_{\mathrm{B}}$, respectively. Clearly, matrices $\tilde{I}_{\mathrm{A}}$ and $\tilde{I}_{\mathrm{B}}$ do not add to the complexity of PMcomp. By restricting the number of base pairs in P^A and P^B by a constant threshold as applied in LocARNA [WRH+07] (cf. Section 3.1), we obtain LocARNA's quartic time and quadratic space complexity. So far, we enhanced PMcomp's (and LocARNA's) model to incorporate the original flexibility from the Sankoff algorithm.

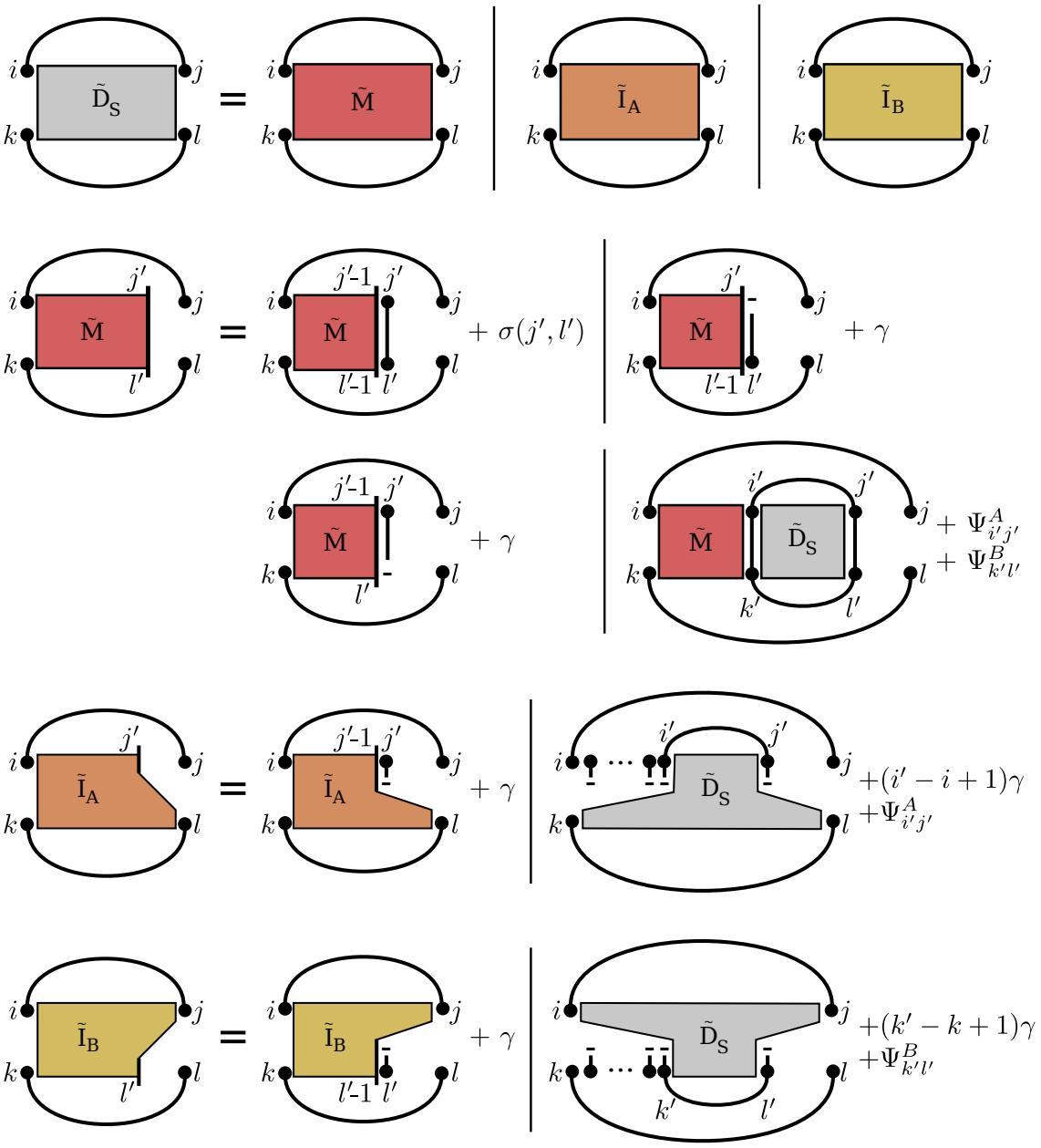


Figure 5.4: **Recursion visualization.** Recursions of the novel lightweight alignment algorithm PARSE. The best alignment of subsequences enclosed by a base pair match is stored in $\tilde{D}_S$. Matrices $\tilde{I}_A$ and $\tilde{I}_B$ cover the 2-loop deletion and insertion case, respectively. The best alignment of the whole input sequences is computed by filling an additional matrix $\tilde{M}$ for the pseudo base pairs ψ_A and ψ_B. Figure adapted from [WOM+15].

$$\tilde{D}_S(ij,kl) = \max \begin{cases} \tilde{M}^{ijkl}(j-1,l-1) \\ \tilde{I}_{\mathrm{A}}^{ijkl}(j-1) \\ \tilde{I}_{\mathrm{B}}^{ijkl}(l-1) \end{cases}$$

$$\tilde{M}^{ijkl}(j',l') = \max \begin{cases} \tilde{M}^{ijkl}(j'-1,l'-1) + \sigma(j',l') \\ \tilde{E}^{ijkl}(j',l') \\ \tilde{F}^{ijkl}(j',l') \\ \max\limits_{\substack{(i',j')\in P^A,\\ (k',l')\in P^B}} \tilde{M}^{ijkl}(i'-1,k'-1) + \tilde{D}_S(i'j',k'l') + \Psi^A_{i'j'} + \Psi^B_{k'l'} \end{cases}$$

$$\tilde{E}^{ijkl}(j',l') = \max \begin{cases} \tilde{E}^{ijkl}(j',l'-1) + \gamma_{\mathrm{base}} \\ \tilde{M}^{ijkl}(j',l'-1) + \beta_{\mathrm{base}} + \gamma_{\mathrm{base}} \end{cases}$$

$$\tilde{F}^{ijkl}(j',l') = \max \begin{cases} \tilde{F}^{ijkl}(j'-1,l') + \gamma_{\mathrm{base}} \\ \tilde{M}^{ijkl}(j'-1,l') + \beta_{\mathrm{base}} + \gamma_{\mathrm{base}} \end{cases}$$

$$\tilde{I}_{\mathrm{A}}^{ijkl}(j') = \max \begin{cases} \tilde{I}_{\mathrm{A}}^{ijkl}(j'-1) + \gamma_{\mathrm{loop}} \\ \max\limits_{(i',j')\in P^A} (i'-i+1)\gamma_{\mathrm{loop}} + \tilde{I}_{\mathrm{A}}^{i'j'kl}(j'-1) + \Psi^A_{i'j'} \\ \max\limits_{(i',j')\in P^A} (i'-i+1)\gamma_{\mathrm{loop}} + \tilde{D}_S(i'j',kl) + \Psi^A_{i'j'} + \beta_{\mathrm{loop}} \end{cases}$$

$$\tilde{I}_{\mathrm{B}}^{ijkl}(l') = \max \begin{cases} \tilde{I}_{\mathrm{B}}^{ijkl}(l'-1) + \gamma_{\mathrm{loop}} \\ \max\limits_{(k',l')\in P^B} (k'-k+1)\gamma_{\mathrm{loop}} + \tilde{I}_{\mathrm{B}}^{ijk'l'}(l'-1) + \Psi^B_{k'l'} \\ \max\limits_{(k',l')\in P^B} (k'-k+1)\gamma_{\mathrm{loop}} + \tilde{D}_S(ij,k'l') + \Psi^B_{k'l'} + \beta_{\mathrm{loop}} \end{cases}$$

$\gamma_{\mathrm{base}}/\gamma_{\mathrm{loop}}$: gap cost for base/loop indel
$\beta_{\mathrm{base}}/\beta_{\mathrm{loop}}$: gap opening cost for base/loop indel

Figure 5.5: **Recursion equations including affine gap costs.** PARSE recursions for computing the best structure alignment triple including affine gap costs. Figure adapted from [WOM+15].

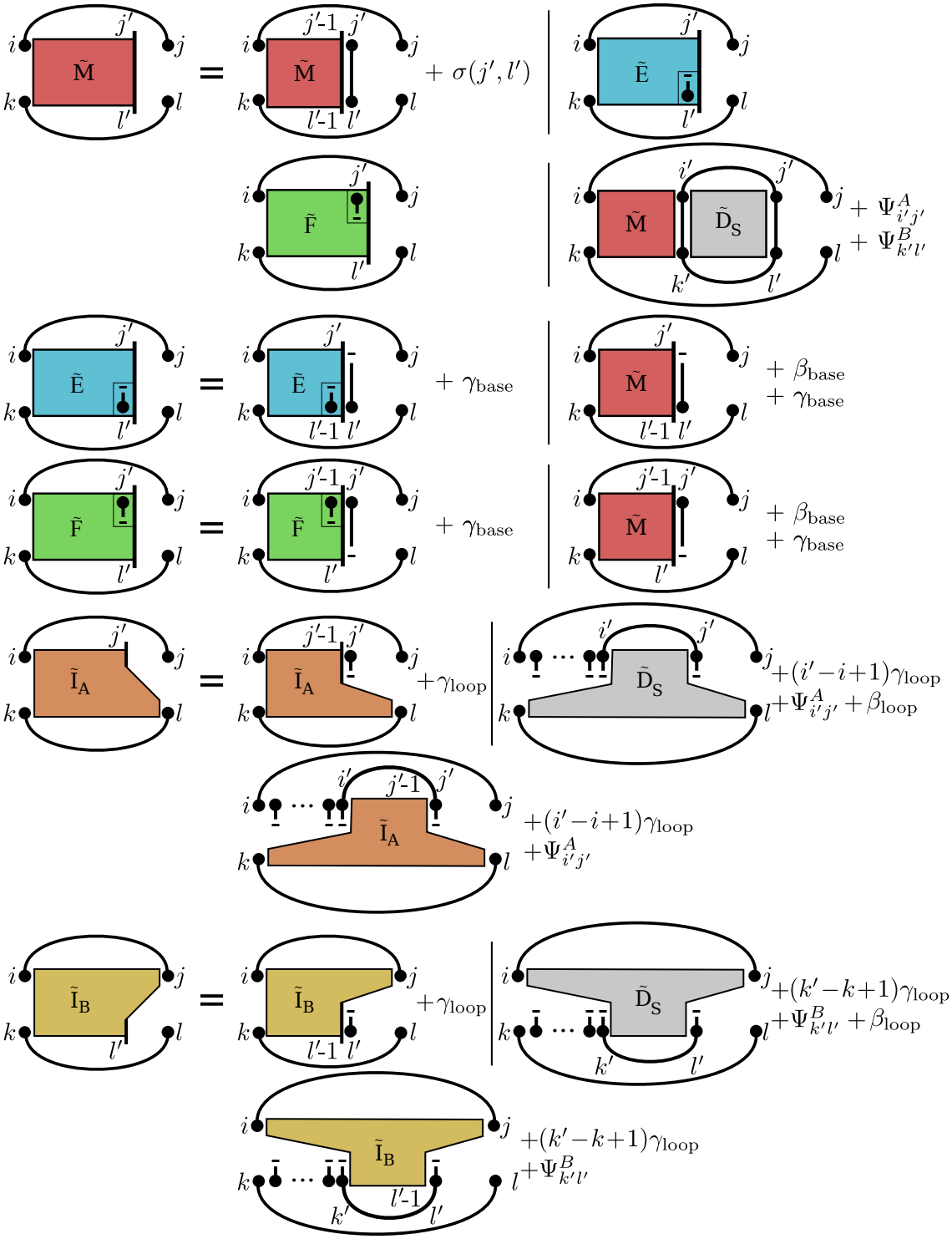


Figure 5.6: **Recursion visualization including affine gap costs.** PARSE recursions for computing the best structure alignment triple including affine gap costs. The visualization for matrix $\tilde{D}_S(ij,kl)$ is analogous to Figure 5.4.

Before we show how to introduce ensemble-based sparsification, we discuss how to incorporate affine gap costs into PARSE. When applying an affine gap cost scheme, the cost of a gap is split into a gap opening cost β and an extension cost γ for each position within the gap. Since PARSE allows to insert or delete entire 2-loops (loop indels) on top of the regular base insertions and deletions (base indels), we distinguish these different events by introducing distinct costs γ_{loop} and γ_{base}. Furthermore, we define β_{loop} and β_{base}, the gap opening costs for loop and base indels, respectively. This distinction makes sense also biologically, since loop indels – unlike base indels – represent the deletion or insertion of structural parts, i.e. the elongation or shortening of stems in the RNA structure. Similar to Gotoh's algorithm [Got82], we incorporate affine gap costs for base indels by introducing the additional matrices $\tilde{E}$ and $\tilde{F}$. The modified recursions are given in Figure 5.5 and a visualization in Figure 5.6. In $\tilde{E}^{ijkl}(j', l')$ base $B_{l'}$ is inserted and in $\tilde{F}^{ijkl}(j', l')$ base $A_{j'}$ is deleted. Then the base insertion and deletion case of the $\tilde{M}$ recursion is replaced by $\tilde{E}^{ijkl}(j', l')$ and $\tilde{F}^{ijkl}(j', l')$. Furthermore, we extend the recursions for $\tilde{I}_{\text{A}}$ and $\tilde{I}_{\text{B}}$ to cover the affine scoring of loop indels. Here, we distinguish the cases that a loop indel is continued ($\tilde{I}_{\text{A}}^{i'j'kl}(j'-1)$) or a new loop indel is started ($\tilde{D}_S(i'j', kl)$). Only in the latter case, we add the gap opening cost β_{base}. Note that this is correct even if $\tilde{D}_S(i'j', kl)$ also implicitly includes the case $\tilde{I}_{\text{A}}^{i'j'kl}(j'-1)$, since we are only interested in the maximal score. Including affine gap costs in PARSE neither increases the runtime nor space complexity.

5.2.2 SPARSE – Sparsifying simultaneous alignment and folding of RNA

In this section, we will use the in-loop probabilities that were introduced in Chapter 3 to sparsify the PARSE algorithm, resulting in the algorithm SPARSE. Similar to ExpaRNA-P, SPARSE does not fill whole dynamic programming matrices, but restricts the recursions to matrix cells that can contribute to a probable solution. All necessary definitions can be found in Chapter 3. A sequence position is denoted by a "prime", e.g. i', and a position in the sparsified matrix by a "bar", e.g. $(\bar{x}, \bar{y})$. By applying fixed thresholds θ_1, θ_2 and θ_3 to the in-loop probabilities, we do not optimize over all structure alignment triples $(\mathcal{A}, R, T)$ but restrict the search to *probable* ones in the structure ensemble (cf. Definition 5.3).

Definition 5.3 (Sparse structure alignment triple)
Given fixed thresholds θ_1, θ_2, and θ_3 (with $\theta_i > 0$ for $i \in \{1, 2, 3\}$), a *sparse* structure alignment triple is a structure alignment triple $(\mathcal{A}, R, T)$, cf. Definition 5.2, that satisfies:

1) for all $(i, j) \in R : \Pr[(i, j)|A] \geq \theta_1$ and for all $(k, l) \in T : \Pr[(k, l)|B] \geq \theta_1$
2) for all $(j', l') \in \mathcal{A}$ with $(i, j) = \text{parent}_R(j')$, $(k, l) = \text{parent}_T(l')$ and j', l' unpaired: $\Pr[j' \in \text{loop}(i, j)|A] \geq \theta_2$ and $\Pr[l' \in \text{loop}(k, l)|B] \geq \theta_2$
3) for all $(i', j') \in R$ with $(i, j) = \text{parent}_R(i', j') : \Pr[(i', j') \in \text{loop}(i, j)|A] \geq \theta_3$ and for all $(k', l') \in T$ with $(k, l) = \text{parent}_T(k', l') : \Pr[(k', l') \in \text{loop}(k, l)|B] \geq \theta_3$

Note that all external bases and base pairs are implicitly filtered with the probability to be external (cf. Definition 3.1). In SPARSE, we restrict the recursions to compute the best sparse structure alignment triple. Thus, for a particular base pair (i, j) only candidate positions (cf. Definition 3.2), i.e. either significant single-stranded positions within (i, j) or positions that are contained in a significant helix of (i, j), need to be considered. The entire row or column can be skipped, if the corresponding sequence position is no candidate. This enables a novel form of sparsification, the ensemble-based sparsification (cf. Chapter 3). Consequently, adjacent matrix positions are not necessarily adjacent on the sequence level. If this is the case, an implicit gap is introduced. But as opposed to ExpaRNA-P, for each skipped row or column, the appropriate gap cost need to be added but an arbitrary number of those implicit gaps can be introduced. For sequence A, the number of gap positions that lie in between a matrix index $\bar{x}$ and a sequence position i' with $\mathsf{pos}^A_{(i,j)}(\bar{x}) < i'$ (see Equation 3.4) is calculated by

$$\mathrm{gaps}^A_{(i,j)}(\bar{x}, i') = (i' - \mathsf{pos}^A_{(i,j)}(\bar{x}) - 1) \tag{5.2}$$

Navigating through the dynamic programming matrices is enabled by the additional data structures defined in Section 3.5. For instance, after considering a base pair match, the valid matrix position $(\bar{x}, \bar{y})$ before the left ends i' and k' of the base pairs can be identified by mat-pos-bef$^{ijkl}(i', k')$, cf. Equation 3.4. Let $i'' = \mathsf{pos}^A_{(i,j)}(\bar{x})$ and $k'' = \mathsf{pos}^B_{(k,l)}(\bar{y})$. Then this means that the entire subsequences $A_{i''..i'}$ and $B_{k''..k'}$ cannot be aligned and have to be inserted or deleted. Thus, if we go from one matrix position to the adjacent one, we have to account for the potentially introduced gaps on the sequence level. In the given example, we consider $\mathrm{gaps}^A_{(i,j)}(\bar{x}, i')$, the number of gaps for sequence A, and $\mathrm{gaps}^B_{(k,l)}(\bar{y}, k')$ for B.

Furthermore, we incorporate the probability cutoffs of the in-loop probabilities into the recursions (see Figure 5.7). First of all, we filter all base pairs in P^A and P^B with threshold θ_1 and use only these constrained sets within the recursions. Furthermore, matrices D_S are computed only for those base pair matches. Filters based on the in-loop probabilities (condition 2) and 3) in Definition 5.3) are applied to the base match and base pair match case.

Previous PMcomp-like implementations as, for example, LocARNA [WRH$^+$07], RAF [DFB08], and FoldAlignM [THG07] allow to predict a base pair in one sequence only if it is covered. Thus, they only keep track of one structure, the consensus structure, for both sequences. But to reasonably apply sparsification for sequence-structure alignment, we need to identify the enclosing loop for each sequence separately as filtering according to the consensus structure does not yield the desired results (see Figure 5.8). In Figure 5.8, if a probable base pair in one sequence could not be predicted as it does not have a corresponding partner in the other sequence (base pair a_2) everything that lies underneath this base pair will be filtered according to a "wrong" parent base pair (a_1) and thus our sparsification technique would not correctly identify the non-candidate positions.

$$D_S(ij,kl) = \max \begin{cases} M^{ijkl}(n_A-1, n_B-1) \\ \quad +\gamma \cdot (\text{gaps}^A_{(i,j)}(n_A-1, j) + \text{gaps}^B_{(k,l)}(n_B-1, l)) \\ I_{\text{A}}^{ijkl}(n_A-1) + \gamma \cdot \text{gaps}^A_{(i,j)}(n_A-1, j) \\ I_{\text{B}}^{ijkl}(n_B-1) + \gamma \cdot \text{gaps}^B_{(k,l)}(n_B-1, l) \end{cases}$$

$$M^{ijkl}(\bar{x},\bar{y}) = \max \begin{cases} \text{if } \Pr[j' \in \text{loop}(i,j)|A] \geq \theta_2 \text{ and } \Pr[l' \in \text{loop}(k,l)|B] \geq \theta_2 \\ \quad M^{ijkl}(\bar{x}-1, \bar{y}-1) + \sigma(j', l') \\ \quad\quad +\gamma \cdot (\text{gaps}^A_{(i,j)}(\bar{x}-1, j') + \text{gaps}^B_{(k,l)}(\bar{y}-1, l')) \\ M^{ijkl}(\bar{x}, \bar{y}-1) + \gamma \cdot \text{gaps}^A_{(i,j)}(\bar{y}-1, l') \\ M^{ijkl}(\bar{x}-1, \bar{y}) + \gamma \cdot \text{gaps}^B_{(k,l)}(\bar{x}-1, j') \\ \text{for all } (i',j') \in P^A, (k',l') \in P^B \\ \quad\quad \Pr[(i',j')|A] \geq \theta_1, \Pr[(i',j') \in \text{loop}(i,j)|A] \geq \theta_3 \text{ and} \\ \quad \text{with } \Pr[(k',l')|B] \geq \theta_1, \Pr[(k',l') \in \text{loop}(k,l)|B] \geq \theta_3 \text{ and} \\ \quad\quad (\bar{p}, \bar{q}) = \text{mat-pos-bef}^{ijkl}(i', k') \\ \quad M^{ijkl}(\bar{p}, \bar{q}) + D_S(i'j', k'l') + \Psi^A_{i'j'} + \Psi^B_{k'l'} \\ \quad\quad +\gamma \cdot (\text{gaps}^A_{(i,j)}(\bar{p}, i') + \text{gaps}^B_{(k,l)}(\bar{q}, k')) \end{cases}$$

$$I_{\text{A}}^{ijkl}(\bar{x}) = \max \begin{cases} I_{\text{A}}^{ijkl}(\bar{x}-1) + \gamma \cdot \text{gaps}^A_{(i,j)}(\bar{x}-1, j') \\ \text{for all } (i',j') \in P^A \\ \quad \text{with } \Pr[(i',j')|A] \geq \theta_1 \text{ and } \Pr[(i',j') \in \text{loop}(i,j)|A] \geq \theta_3 \\ (i'-i+1)\gamma + D_S(i'j', kl) + \Psi^A_{i'j'} \end{cases}$$

$$I_{\text{B}}^{ijkl}(\bar{y}) = \max \begin{cases} I_{\text{B}}^{ijkl}(\bar{y}-1) + \gamma \cdot \text{gaps}^B_{(k,l)}(\bar{y}-1, l') \\ \text{for all } (k',l') \in P^B \\ \quad \text{with } \Pr[(k',l')|A] \geq \theta_1 \text{ and } \Pr[(k',l') \in \text{loop}(k,l)|B] \geq \theta_3 \\ (k'-k+1)\gamma + D_S(ij, k'l') + \Psi^B_{k'l'} \end{cases}$$

$$j' = \mathsf{pos}^A_{(i,j)}(\bar{x}),\ l' = \mathsf{pos}^B_{(k,l)}(\bar{y})$$
$$n_A = \left|\mathsf{pos}^A_{(i,j)}\right|,\ n_B = \left|\mathsf{pos}^B_{(k,l)}\right|$$

Figure 5.7: **Recursion equations for SPARSE.** SPARSE recursions for computing the best sparse structure alignment triple on sparsified matrices.

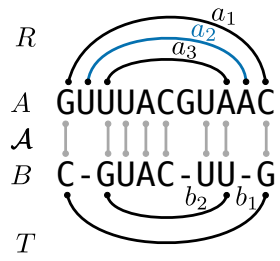


Figure 5.8: **Sparsification needs loop indels.** Possible base pairs for A are a_1, a_2 and a_3, and for B b_1 and b_2. If a_2 is probable, the probability that a_3 is contained in the loop closed by a_1 in A is low. Thus, to reasonably apply sparsification, a_2 has to be predicted without being matched to a base pair in B as a_3 is much more likely to be in the loop closed by a_2 due to stacking effects. Figure adapted from [WOM+15].

Due to the filtering by fixed thresholds θ_1, θ_2, and θ_3, the dynamic programming matrices become sparse and thus our novel tool achieves quadratic time and space complexity (cf. Theorem 5.1).

Theorem 5.1 *SPARSE finds the optimal sparse structure alignment triple in $O(n^2)$ time and space.*

Proof. Analogously to the argument in ExpaRNA-P, there are $O(n^2)$ combinations i, j, k, l, j', l' satisfying the candidate condition from Definition 3.2. Thus, the time and space complexity of computing all entries $M^{ijkl}(\bar{x},\bar{y})$, $I_{\mathrm{A}}^{ijkl}(\bar{x})$, $I_{\mathrm{B}}^{ijkl}(\bar{y})$ is $O(n^2)$. Due to Proposition 3.1 (linear number of base pairs), matrix D_S has only $O(n^2)$ entries and matrix entries from all matrices can be computed in constant time. □

Albeit this does not affect the complexity, we can apply another optimization strategy that was already utilized in LocARNA. During the computation of M matrices, overlapping subsequences are considered multiple times if the left ends of the base pairs coincide. Thus matrices M^{ijkl} and $M^{i'j'k'l'}$ can be combined if $i = i'$ and $j = j'$. Here, filling one M matrix suffices for each i k combination that combines the computation of all M^{ijkl} for all j and l. A position is considered if it is a candidate in any of these matrices. If a position is a candidate for different matrices that can be combined, we safe computation time in practice. Note that applying this technique guarantees that all sparse structure alignment triples are considered during the computation, but does not necessarily prohibit non-sparse solutions. Due to the combined computation of different M matrices, a base can be matched that is a candidate for an (unpredicted) base pair (i,j) and even if it is no candidate for (i,j') can still be matched within this base pair. It usually suffices in practice to find some (not necessarily sparse) triple whose score is at least as high as the score of any sparse triple.

In ExpaRNA-P (Chapter 4), we do not use this optimization strategy but compute a individual matrix for each base pair match. ExpaRNA-P identifies sequence-structure motifs, termed EPMs, in entire Boltzmann-distributed structure ensembles of two RNAs. These are used in a second step to speed up LocARNA, an algorithm for computing sequence-structure alignments of RNAs. Crucially, we need to make sure to identify probable EPMs that can serve as

reliable anchors for a subsequent LocARNA run. By filling one matrix per base pair match, we assure that each matched position fulfills the filter according to the in-loop probabilities of the predicted parent base pair and is thus probable to occur within the predicted structure.

5.2.3 Multiple alignment

In order to efficiently construct a multiple alignment based on the pairwise SPARSE algorithm, we apply the widely used progressive strategy. First, we create a guide tree based on pairwise alignments computed by SPARSE. In the progressive phase, we compute consensus dot plots for subalignments with RNAalifold [HFS02, BHW$^+$08] and use SPARSE to align the subalignments. We describe the procedure for aligning N sequences with maximal length n. Since all in-loop probabilities and additional data structures (cf. Chapter 3) can be precomputed separately for each input sequence, in total $O(Nn^3)$ time is needed. All $O(N^2)$ many pairwise alignments between all pairs of the input sequences can be calculated in $O(N^2n^2)$, from which the guide tree is constructed by UPGMA [GM07] in $O(N^2)$ time. In principle, RNAalifold can be seen as an extension of the McCaskill algorithm [McC90] to multiple sequences and computes the mfe structure and partition function for a given multiple sequence alignment. For the progressive phase, we extended RNAalifold to calculate in addition the in-loop probabilities without changing the time complexity of $O(n^3)$. In each step during the progressive phase, a (pairwise) alignment of two subalignments is computed in $O(n^2)$ time. The progressive phase takes $O(Nn^3 + Nn^2) \in O(Nn^3)$ time in total as there are $N - 1$ steps necessary to compute the whole multiple alignment. The final multiple alignment pipeline requires $O(Nn^3 + N^2n^2)$ time whereas the corresponding pipeline based on LocARNA needs $O(N^2n^4)$ time.

5.3 Results

We embedded our C++ implementation of SPARSE in the LocARNA framework [WRH$^+$07]. We use from BRAliBase 2.1 [WMS06, GWW05] the pairwise benchmark set k2 and the three-way alignment set k3. For SPARSE we used the following parameter configuration: $\theta_1 = 0.001$, $\theta_2 = 0.00005$, $\theta_3 = 0.0001$, $\beta_{\text{base}} = -900$, $\gamma_{\text{base}} = -3500$, $\beta_{\text{loop}} = -900$ and $\gamma_{\text{loop}} = -350$.[2]

5.3.1 Comparison with other tools

To evaluate the performance of SPARSE, we benchmarked it against LocARNA [WRH$^+$07] and RAF [DFB08] with respect to runtime and alignment quality.[3] Table 5.1 shows that SPARSE and RAF achieve a considerable speed up over LocARNA of about 4 on benchmark

[2]The parameters were selected ad-hoc without parameter learning.
[3]We used default parameters for LocARNA and RAF.

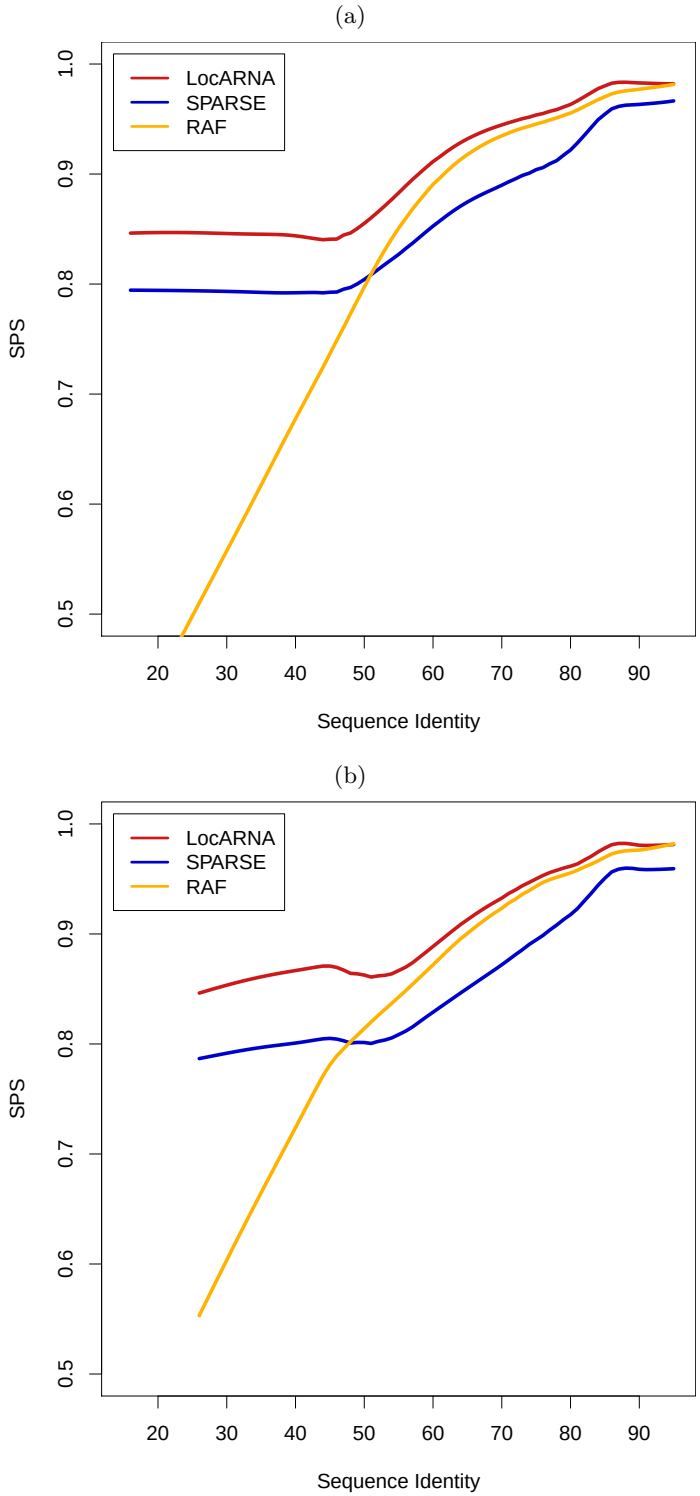


Figure 5.9: **Comparison with sequence-structure alignment methods.** The alignment quality measured by the SPS score is displayed for different sequence identity ranges as lowess curves [Cle81] for benchmark set (a) k2 and (b) k3. Figure adapted from [WOM+15].

Table 5.1: **Comparison of RNA alignment methods.** Total runtime and speed up of pairwise alignments across BRAliBase 2.1 set k2. The speed up is measured relative to LocARNA. Table taken from [WOM+15].

Tool	Total time (s)	Mean Time (s)	Speedup (vs. LocARNA)
LocARNA	13400	1.49	1.0
SPARSE	3600	0.40	3.7
RAF	3200	0.36	4.2

set k2. To measure the alignment quality, we compare for each instance the derived alignment with the known reference alignment using the tool compalignp [WMS06]. If the computed alignment perfectly matches the reference alignment, a SPS score of 1 is assigned whereas alignments without any correspondence receive a SPS score of 0. We show in Figure 5.9a the dependency of the sequence identity on the SPS score, visualized by lowess curves for the different tools [Cle81]. Clearly, RAF's and SPARSE's curve show a completely different behavior: whereas SPARSE maintains a high alignment quality over the entire range of sequence identities, RAF's performance breaks down for lower sequence identities. This dramatic drop for RNAs with sequence identities below 60% resembles pure sequence alignment methods and might be a consequence of the strong sequence-based heuristics applied in RAF. The quality difference between LocARNA and SPARSE remains largely constant and both tools achieve a high alignment quality even for the "hard" instances with low sequence identity. To test the performance of our multiple alignment pipeline, we additionally benchmark the tools on benchmark set k3. The evaluation in Figure 5.9b show that all tools behave comparably on multiple alignment instances.

To summarize, it should be noted that RAF and SPARSE show a huge difference regarding the alignment quality while obtaining a similar speed up.

5.3.2 Flexible structure prediction of SPARSE improves folding accuracy

In a second experiment, we evaluate the impact of the more flexible structure prediction model in SPARSE for benchmark set k2. Recall that in SPARSE, we integrate the full Sankoff model by allowing loop insertions and deletions and thus predicting individual but compatible structures for the two input sequences. All previous lightweight approaches like PMcomp or LocARNA compute only a single consensus structure. The accuracy of each predicted structure is measured by Matthews correlation coefficient (MCC) [Mat75] by comparing it with the constrained folded reference consensus structure of the corresponding sequence from Rfam [BDE+13]. We compare SPARSE and LocARNA to isolate the effect of enhanced flexibility and sparsification since these two tools behave as similar as possible aside from that. The distribution of MCC

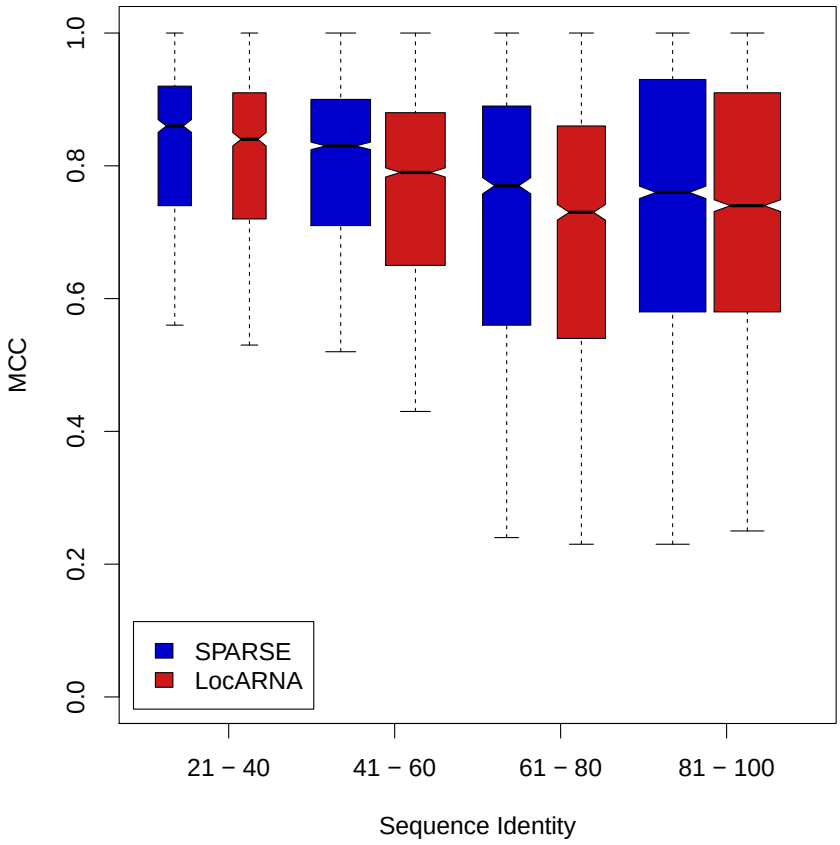


Figure 5.10: **Accuracy of structure prediction.** Quality of structure prediction measured by Matthews correlation coefficient (MCC) [Mat75] on benchmark set k2 for SPARSE and LocARNA for different sequence identity ranges. The whiskers of the boxplots are extended up to one interquartile range from the boxes. As the notches of the two boxplots do not overlap, there is strong evidence for an improved structure prediction ability [CCKT83] of SPARSE for all sequence identity ranges. Figure adapted from [WOM+15].

values is visualized as boxplot for different sequence identity regions in Figure 5.10. Compared with LocARNA, the benefit of SPARSE's flexible model is reflected in a strong evidence for improved structure prediction quality, visualized by the non-overlapping notches of the boxplots [CCKT83]. Notably, this is independent from the sequence identity of the benchmark instance.

As a concrete example for such a case, we compare the alignment and folding of two RNAs from family gcvT by LocARNA and SPARSE (see Figure 5.11). This example illustrates the benefits of incorporating loop deletions and insertion. Since LocARNA disallows loop deletions and insertions, it cannot predict structure in deleted regions; consequently, it predicts large unstable loops. In contrast, SPARSE supports loop deletions (represented by '_' in the alignment) and thus predicts a long stem in sequence A, which is deleted in sequence B. Thereby, the remaining parts can also form a stable structure with short loop regions. This behavior is reflected by the

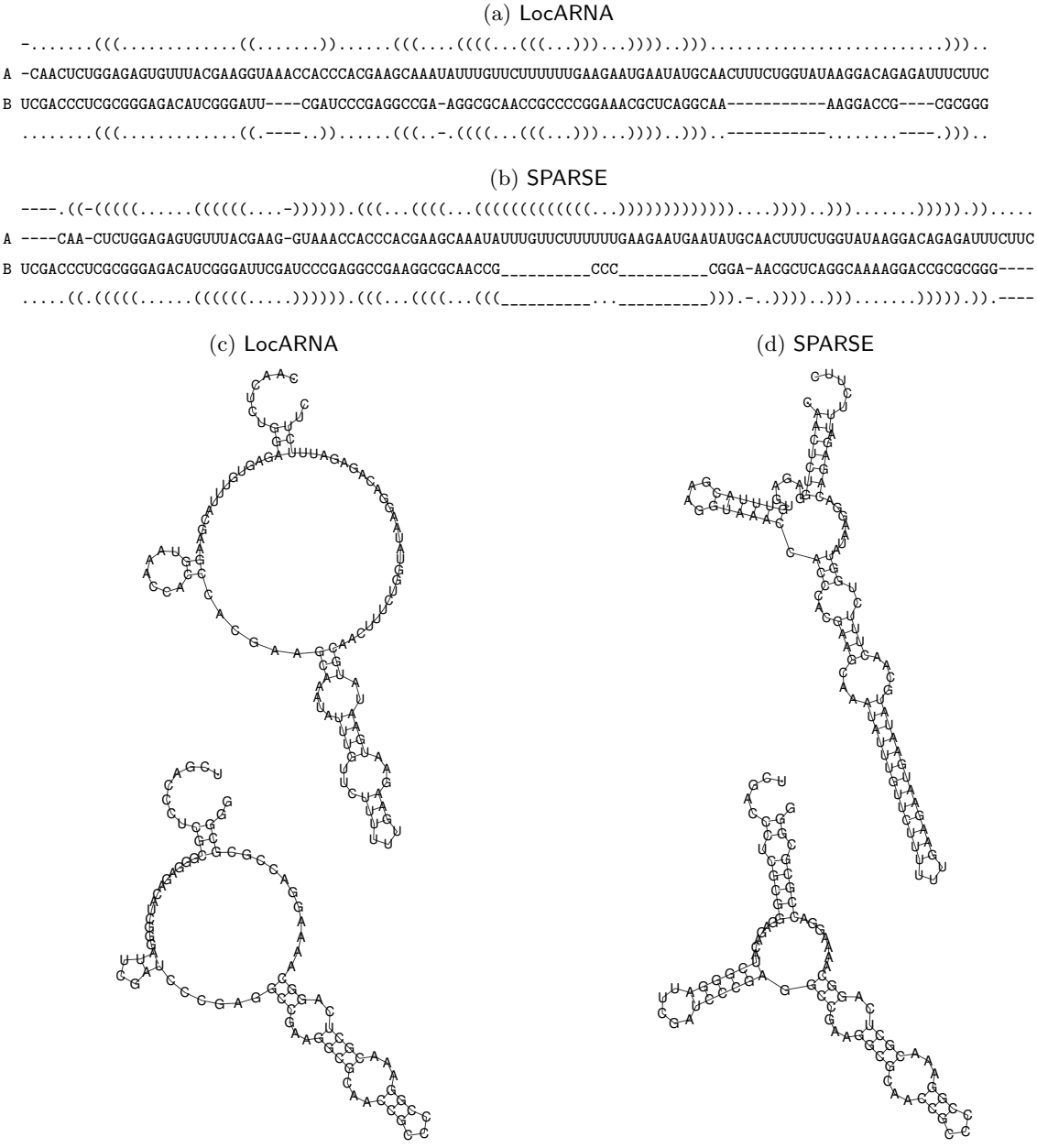


Figure 5.11: **Example from gcvT family where SPARSE improves LocARNA's structure prediction.** The sequence-structure alignments computed by LocARNA and SPARSE are shown in (a) and (b). Subfigures (c) and (d) visualize the predicted structures projected on both input sequences by RNAplot [LBHZS+11]. Since LocARNA cannot predict structure in deleted regions, large unpaired regions are predicted in the multiloop, which destabilize the structures. The more flexible model in SPARSE allows loop deletions and insertions (represented by '_' in the alignment) and thus can align stems of varying length; in this example, the stems at the bottom. This results in smaller loops and thus more stable structures. Figure adapted from [WOM+15].

MCC prediction quality scores: LocARNA achieves a MCC of 0.51, which SPARSE improves to 0.94.

5.4 Discussion

We presented PARSE, the first algorithm that combines the original flexibility of Sankoff's algorithm with PMcomp's lightweight energy model. Integrating our novel ensemble-based sparsification in PARSE, results in the sparsified variant SPARSE. The key is to identify likely structural features based on the probability that a base or base pair occurs in a specific loop within the structure ensemble. To apply reasonable filtering, the integration of loop deletions and insertions into the structure prediction model is crucial. Ensemble-based sparsification cannot be directly implemented in previous PMcomp-like implementations like LocARNA, RAF and FoldAlignM as these tools only keep track of a single predicted structure, the consensus structure, for both sequences. SPARSE, on the other hand, predicts individual but compatible structures for the input sequences, which is an important prerequisite to calculate the in-loop probabilities for ensemble-based sparsification. SPARSE is the first tool that solves the simultaneous alignment and folding problem in quadratic time without the need to employ sequence-based heuristics that could compromise the alignment quality. This makes SPARSE's runtime equivalent to the currently fastest tool RAF, which also manifests itself in an overall speed up of around 4 over LocARNA for both tools on the pairwise benchmark set k2 of BRAliBase 2.1. Unlike RAF that uses sequence-based heuristics to speed up computation, SPARSE achieves high alignment quality for all sequence identity regions with merely a constant offset compared with LocARNA. The superiority of SPARSE over RAF is also evident for the benchmark set of three-way alignments in BRAliBase 2.1. Thus, we also validated that the results for pairwise alignment can be transfered to multiple sequence-structure alignment. Furthermore, we demonstrated that the enhanced structure prediction model of SPARSE significantly improves the structure prediction over LocARNA. Notably, this is the case for all sequence identity regions of the benchmark instance. We provided a concrete example for RNA family gcvT that clearly shows the advantages of predicting individual but compatible structures for the input sequences. Since SPARSE supports loop insertions and deletions, stems of different sizes can be matched and the predicted structures are overall more stable. This is also reflected in a drastically improved structure prediction quality measured by Matthews correlation coefficient (MCC).

CHAPTER 6

General extension for sparsification in ADP

In this chapter, we introduce a novel approach that combines for the first time the well-established concept of dynamic programming (DP) with the relatively recent sparsification technique. For that, we propose a novel sparsification operator that is not restricted to a specific recursive structure nor objective function. The description in this chapter is based on [MSZ11].

DP is a widely used technique in bioinformatics and other research areas to efficiently compute solutions to problems associated with a search space of exponential size. If the optimal solution can be computed from optimal solutions of subproblems, i.e. Bellman's principle of optimality holds (cf. Section 1.3.1), DP can be applied to find the optimal solution in polynomial time. A DP algorithm is defined by recursion equations and a scoring scheme that defines how solutions are evaluated. In a typical DP algorithm, the scoring scheme is embedded in the recursion. In the case of sequence alignment, the recursions (defined by Equation 2.5) already include the scoring scheme where each base match is scored by $\sigma(i, k)$ and each gap by γ.

Although in theory, each polynomial time algorithm is considered efficient, polynomial time algorithms can still be too slow in practice, especially for large problem sizes. Sparsification was applied to numerous algorithms in the field of RNA bioinformatics to achieve an even faster computation, both in theory and practice without compromising optimality [WZZU07, ZUGVWS08, ZUGVWS10, BTZZU11, SMW$^+$10, MSW$^+$10]. The basic idea of sparsification is to identify those parts of the search space that can be ignored since they cannot contribute to an optimal solution anymore (see 1.3.2 for a detailed description). Note that in our novel ensemble-based sparsification technique introduced in Chapter 3, we discard subsolutions that are *unlikely* in the structure ensembles of the RNA sequences.

Algebraic dynamic programming (ADP) is a formalism that allows defining DP algorithms over sequence data in a simple, declarative, and formally precise way [GMS04]. ADP sepa-

rates the search space defined by the recursive structure and the scoring scheme. This is a main feature as the algorithms defined in ADP can be easily re-used by modifying the scoring scheme without having to change the recursive structure. As there is no explicit separation in a usual DP algorithm, the recursions itself have to be modified if a different scoring scheme is applied. As ADP is formalized on a higher level, the need for typically error-prone subscripts in the definition of the recursions is eliminated. ADP is easy to use as there also exist compilers [GS06, SJG11] that yield automatically generated implementations for ADP programs. In addition, the ADPfusion library provides efficient code directly in the functional programming language Haskell [HzS12]. As ADP is very general, it has been used to develop tools for a variety of different tasks like RNA shape analysis (RNAshapes [SVR$^+$06]), prediction of RNA pseudoknot structure (pknotsRG [RSG07]), or the prediction of microRNA/target duplexes (RNAhybrid [RSHG04]).

Contributions While it is intuitively clear that all sparsification approaches mentioned above are somehow related, they have not been characterized in term of a common framework that highlights their similarities and differences. For the first time, we combine the two techniques ADP and sparsification. For that, we define a general sparsification operator that is incorporated into the ADP framework. This leads to a more general understanding of the entire concept of sparsification and allows to quickly implement sparsified variants of DP algorithms formulated in ADP. So far, sparsification has only been applied to problems with a simple choice function based on minimization or maximization. We show that our novel sparsification operator generalizes effortlessly to advanced choice functions, such as the enumeration of suboptimal solutions. We analyze the number of split points that need to be examined during the recursive computation for random RNA sequences of different length. Crucially, we show that the time savings of the sparsified variants increase with the input size, such as the length of the input sequence or the number of enumerated suboptimal solutions. To keep the presentation simple, we show how to apply the novel sparsification operator to the Nussinov algorithm. It can be used analogously for more complex recursions. We describe the subsequent algorithms for a sequence A of length n.

6.1 A quick overview on ADP

DP algorithms are usually described with recursion equations. Those equations contain all essential aspects of the program including the recursive structure of the problem decomposition, the scoring scheme, and the question which sub-results should be tabulated. ADP is an alternative way to describe DP algorithms over sequence data, which explicitly separates all those aspects. Thus, each ADP program consists of two ingredients: the grammar and the algebra. The grammar defines the recursive structure and the algebra the scoring scheme.

We introduce the concepts of ADP by showing an example ADP program for the Nussinov algorithm [NPGK78]. A detailed description of ADP can be found in [GMS04].

An ADP program for the Nussinov algorithm (cf. Equation 2.1) would consist of these four grammar rules:

$$N' \rightarrow \underset{\epsilon}{\text{nil}} \mid \underset{a}{\text{single}} \mid \underset{N'\ a}{\text{unpaired}} \mid \underset{N'\ a\ N'\ \hat{a}}{\text{paired}} \quad \ldots h \tag{6.1}$$

representing the two base cases and the two recursive cases. The grammar is similar to a context free grammar (CFG) but with trees on the right hand side of the rules. If only the leaves of the trees are considered, a normal context free grammar is obtained. In the example, the grammar has a nonterminal symbol N' and terminal symbols a and $\hat{a}$ representing arbitrary bases such that $(a, \hat{a})$ is a valid base pair.

The choice function h describes the optimization criterion, i.e. which of the parses are the solution(s). The algebra of the ADP program evaluates a parse and thus represents the scoring scheme. For the Nussinov algorithm, the algebra just counts the number of base pairs that are introduced in the "paired" case and the choice function h maximizes the score:

$$\begin{aligned} &\text{nil}() = 0 && \text{unpaired}(x, a) = x && h(X) = \max X \\ &\text{single}(a) = 0 && \text{paired}(x, a, y, \hat{a}) = x + y + 1 \end{aligned} \tag{6.2}$$

The inner nodes of the trees are functions of the algebra. During the run of an ADP program, the input sequence is parsed as in a usual CFG. But since the right hand side of the grammar rules are trees, each derivation constructs a tree instead of a string by recursively replacing each nonterminal by the subtree derived by it. The input sequence "CAG" has, for example, different parses (see Figure 6.1) representing the structure without base pairs and the one with base pair C−G. Each such tree can be interpreted as a term whose evaluation yields some score.[1] In this example, the first tree evaluates to 0 and the second one to 1 since the trees represent structures with 0 and 1 base pair, respectively. The choice function h takes as an argument a set X, which represents a set of parses (or more precisely a multiset of their scores) and returns the optimal score(s). For the algebra in Equation 6.2, the choice function selects the maximal score. In most cases the result is a singleton set, but ADP programs can also, for example, enumerate all possible parses or pick the best 10.

When the choice function is applied at the very end, after the parser has constructed all possible parses, the ADP program behaves like a simple generate and test program that enumerates all possible structures and then picks the one with maximum number of base pairs. But as

[1] Note that this two-step procedure is crucial for the separation of the search space and the evaluation and cannot be modeled by a context free grammar.

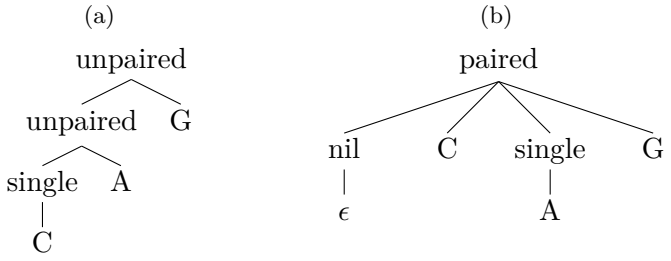


Figure 6.1: **Example for parses.** Two parses of the input "CAG" for the grammar given in Equation 6.1. The parses in (a) and (b) represent a structure without base pairs and with base pair C−G, respectively. Evaluation with the algebra in Equation 6.2 yields a score of 0 and 1 for the parses in (a) and (b), respectively. Figure taken from [MSZ11].

CFGs can be efficiently parsed by the Cocke-Younger-Kasami algorithm (CYK), this can be turned into efficient dynamic programming by interleaving a CYK like grammar parser with the application of the choice function. This is indicated by adding "$\ldots h$" to the right-hand side of a grammar rule. More precisely, the CYK parser runs as usual recursively computing for all subsequences from some i to some j and each nonterminal X the set of all possible parses. But instead of storing in a table entry $X(i,j)$ the set of all parses, the parses are first evaluated to their score and then the choice function is applied such that only the (multisets of) optimal scores need to be stored in the table. Assume, for example, that "CAG" is a part of some longer input sequence. Then only the second possible parse for "CAG" (cf. Figure 6.1b) would be considered when recursively constructing parses for any larger parts of the input, as its score is the maximum among the two possible parses of the sequence.

Separating the recursion equations (grammar), and the scoring scheme (algebra), offers a great advantage as different algebras can be defined without having to adjust the recursion equations. Thus, ADP is very modular. For the Nussinov example, an additional algebra can be defined by

$$\begin{array}{lll} \text{nil'}() = \text{''} & \text{unpaired'}(x,a) = x + \text{'.'} & h'(X) = X \\ \text{single'}(a) = \text{'.'} & \text{paired'}(x,a,y,\hat{a}) = x + \text{'('} + y + \text{')'} & \end{array} \tag{6.3}$$

(where + denotes string concatenation) to enumerate all possible structures as dot bracket strings instead of computing the structure with maximum number of base pairs. In this example, the choice function does not select the maximal score, but considers the scores of all parses to enumerate all structures. In our example, the two parses in Figure 6.1 would evaluate to '...' and '(.)', representing the two possible structures for the sequence CAG.

6.2 Sparsified variants of the Nussinov algorithm

Before we introduce our novel sparsification operator, we describe how sparsification can be applied to the Nussinov algorithm. In general, sparsification allows to completely ignore regions of the search space if it is guaranteed to find a solution outside these regions. Thus, by skipping non relevant parts of the search space, the runtime of an algorithm is improved *without* sacrificing optimality. For that, we introduce an alternative recursive decomposition that enables sparsification for Nussinov's algorithm (Figure 6.2). According to this decompositions, any RNA structure (represented by N) can either be split into two parts (represented by S) or is a closed structure (represented by C) that consists either of a single base or an outermost base pair with an arbitrary structure below.

Recall that we evaluate for each nonterminal for subsequence $A_{i..j}$ the set of parses, evaluate them according to the evaluation algebra and select with respect to the choice function. For the ADP formulation of Nussinov's algorithm (see Figures 6.2a,b), $X(i,j) = \{x_{ij}\}$ thus contains the best score x_{ij} that is obtained when $A_{i..j}$ is parsed by nonterminal $X \in \{N, S, C\}$.[2] Figure 6.2c shows the corresponding DP recursions.

In the case of Nussinov's algorithm, the basic idea of sparsification is to reduce the number of split points that need to be examined in grammar rule S (cf. Figure 6.2a). Crucially, this will be accomplished without sacrificing optimality.

Definition 6.1 ((Optimal) split point [WZZU07, BTZZU11])
Let $N(i,j) = \{n_{ij}\}$, $S(i,j) = \{s_{ij}\}$ and $C(i,j) = \{c_{ij}\}$ be defined by the grammar and algebra in Figures 6.2a,b. A *split point* q ($i < q \leq j$) for $A_{i..j}$ partitions the subsequence $A_{i..j}$ into two subsequences $A_{i..q-1}$ and $A_{q..j}$. A split point is called *optimal* if $s_{ij} = n_{iq-1} + c_{qj}$.

To give a basic idea about sparsification, we give a simple, straightforward example. A naive evaluation of the recursions defined in Figure 6.2 would consider all possible split points q for each subsequence $A_{i..j}$. But many of these split points cannot contribute to the solution as $a = A_q$ and $\hat{a} = A_j$ have to form a valid base pair in grammar rule C. This suggests that a lot of split points do not have to be inspected without sacrificing the optimal solution. Though this gives a considerable speedup *in practice*, the theoretical complexity is not reduced. We review in the following two main sparsification criteria – namely the OCT and STEP criterion – for RNA structure prediction that were introduced in [WZZU07, BTZZU11]. Both criteria lower the time complexity by reducing the number of split points that need to be considered in grammar rule S.

[2] Note that ADP uses the set notation to keep the description as universal as possible and to allow representing choice functions other than maximization or minimization (cf. Equation 6.3).

(a)

$$
\begin{array}{lcl}
N & \to & \text{nil} \mid C \mid S \quad \ldots h \\
 & & \quad | \\
 & & \quad \epsilon \\
S & \to & \text{split} \quad \ldots h \\
 & & \quad N \quad C \\
C & \to & \text{single} \mid \text{pair} \quad \ldots h \\
 & & \quad | \qquad\quad a \quad N \quad \hat{a} \\
 & & \quad a
\end{array}
$$

(b)

$$
\begin{aligned}
\text{nil}() &= 0 \\
\text{split}(x, y) &= x + y \\
\text{single}(a) &= 0 \\
\text{pair}(a, x, \hat{a}) &= x + 1 \\
h(X) &= \max X
\end{aligned}
$$

(c)

$$
N(i,j) = \max \begin{cases} S(i,j) \\ C(i,j) \end{cases}
$$

$$
S(i,j) = \max_{\substack{q \text{ with} \\ i<q\leq j}} N(i, q-1) + C(q, j)
$$

$$
C(i,j) = \begin{cases} N(i+1, j-1) + 1 & \text{if } (i,j) \text{ valid base pair} \\ -\infty & \text{otherwise} \end{cases}
$$

Initialization:

$$
\left.\begin{array}{l} S(j, j-1) = -\infty \\ C(j, j-1) = 0 \end{array}\right\} \text{ for all } 1 < j \leq n
$$

$$
\left.\begin{array}{l} S(j, j) = -\infty \\ C(j, j) = 0 \end{array}\right\} \text{ for all } 1 \leq j \leq n
$$

Figure 6.2: **Nussinov variant suitable for sparsification.** ADP formulation with the grammar displayed in (a) and the algebra in (b). Any RNA structure (represented by N) can either be split into two parts (represented by S) or is a closed structure (represented by C) that consists either of a single base or an outermost base pair with an arbitrary structure below. (c) Corresponding dynamic programming recursions including initialization. Note that the single base case is covered implicitely in $S(i, j)$ with split point $q = j$, where $C(j, j)$ is assigned a score of 0 (no base pairs) in the initialization. Figure adapted from [MSZ11].

6.2.1 OCT sparsification

We first study the OCT criterion that introduces an additional constraint into the recursive computation.

Definition 6.2 (OCT criterion [WZZU07, BTZZU11])
A subsequence $A_{i..j}$ fulfills the optimally co-terminus (OCT) criterion if $i = j$ or if every optimal folding of it contains the base pair (i, j), i.e. for $C(i, j) = \{c_{ij}\}$ and $S(i, j) = \{s_{ij}\}$ (defined by Figures 6.2a,b) it holds $c_{ij} > s_{ij}$.

One can show that there is an optimal split point q for $A_{i..j}$ such that $A_{q..j}$ fulfills the OCT criterion. Here we just give an intuitive idea why this holds; the full proof can be found in [WZZU07, BTZZU11]. Suppose q is an optimal split point for $A_{i..j}$. If there exist another optimal split point $q' > q$ for $A_{i..j}$, it is also an optimal split point for $A_{q..j}$. Consequently, $A_{q..j}$ does not fulfill the OCT criterion. In this case we do not need to consider the split point q in S. If we however pick q to be the rightmost optimal split point, by definition there exists no optimal split point $q' > q$ for $A_{i..j}$. One can show that this holds also for $A_{q..j}$ such that $A_{q..j}$ fulfills the OCT criterion. By that, we know that there is at least one optimal split point where the right fragment fulfills the OCT criterion and it suffices to check only those in S.

6.2.2 OCT-STEP sparsification

The number of split points that need to be considered in S can be even further reduced by introducing the STEP criterion.

Definition 6.3 (STEP criterion [BTZZU11])
Let $N(i, j) = \{n_{ij}\}$ be defined by the ADP program in Figures 6.2a,b. The subsequence $A_{i..j}$ fulfills the STEP criterion, if $n_{ij} > n_{ii} + n_{i+1j}$. This is fulfilled if none of the optimal foldings allow the split point $i + 1$, or equivalently, if in every optimal folding base i is paired.

One can show that there is an optimal split point q for $A_{i..j}$ such that either $q = i + 1$, or $A_{i..q-1}$ fulfills the STEP and $A_{q..j}$ the OCT criterion [BTZZU11].

6.3 A general extension for sparsification in ADP

After we have introduced all necessary basics for ADP and exemplified the principle of sparsification for RNA structure prediction, we describe how to incorporate sparsification into the ADP framework. Up to now, sparsification has not been discussed as a well defined concept but merely a collection of examples. The common basis of all those approaches [WZZU07, ZUGVWS08, SMW+10, MSW+10] is to introduce specific tests at some places in the recursion

to determine whether the currently computed value is a *candidate*. For (some of the) subsequent computations then only candidates need to be considered whereas other entries can be safely ignored without affecting the correctness of the recursion. The test for candidacy depends on the respective algorithm (e.g. the OCT and STEP criteria for RNA structure prediction in Section 6.2). We propose the following general extension of ADP that is neither limited to any fixed such criterion nor to a specific choice function h. Intuitively, $C'(i,j)$ in Definition 6.4 is the part of $C(i,j)$ that can be safely removed according to a certain sparsification criterion without compromising optimality, i.e. $C'(i,j)$ represents all non-candidate entries.

Definition 6.4 (Sparsification operator)
Given an ADP program with nonterminals S and C and a choice function h. $C(i,j)$ and $S(i,j)$ contain the result of parsing $A_{i..j}$ by C and S, respectively, after scoring with the algebra and selecting by the choice function h. Then, we define the novel sparsification operator $\setminus_h$ by

$$[C \setminus_h S](i,j) := \{C(i,j) - C'(i,j) \mid h(C(i,j) \uplus S(i,j)) = h((C(i,j) - C'(i,j)) \uplus S(i,j)) \text{ and } |C'(i,j)| \text{ is maximal}\}.$$

$C(i,j) \uplus S(i,j)$ creates a multiset, i.e. if some entry occurs once in both $S(i,j)$ and $C(i,j)$ it occurs twice in the result.

A new nonterminal C^c that represents only the candidates among the possible parses of C can be obtained with a grammar rule $C^c \to C \setminus_h S$. If h encodes maximization (as for example for the Nussinov algorithm), then – for some subsequence $A_{i..j}$ – C^c has a parse only if parsing $A_{i..j}$ by C evaluates better than parsing $A_{i..j}$ by S. We will show in Section 6.3.1 that C^c represents exactly the subsequences that fulfill the OCT criterion (with the definitions given in Figure 6.3). But our novel sparsification operator also generalizes nicely to other choice functions and also result sets of size greater one, as we will see in Section 6.3.3.

Note that there might be more than one set that satisfies the criterion for $C'(i,j)$ in Definition 6.4. In this case, we assume that $C'(i,j)$ is selected arbitrarily among these sets. Technically, also less strict forms of sparsification are conceivable where $|C'(i,j)|$ is not maximal. In typical cases, however, it is preferable to exploit the full extend of reduction that the sparsification technique offers and thus to filter as strongly as possible.

6.3.1 Application to sparsified variants of Nussinov's algorithm

In the following we show how the novel sparsification operator can be utilized to incorporate the OCT and STEP criteria in the Nussinov algorithm (cf. Section 6.2). The bottleneck of the algorithm lies in the computation of S where all split points need to be considered. Introducing the above mentioned criteria significantly reduces the number of split points.

(a)

$N \rightarrow \text{nil}_{\epsilon} \mid C^c \mid S \quad \ldots h$

$S \rightarrow \text{split}_{N\ C^c} \quad \ldots h$

$C \rightarrow \text{single}_{a} \mid \text{pair}_{a\ \mathrm{N}\ \hat{a}} \quad \ldots h$

$C^c \rightarrow C \setminus_h S$

(b)

$$\begin{aligned} \text{nil}() &= 0 \\ \text{split}(x, y) &= x + y \\ \text{single}(a) &= 0 \\ \text{pair}(a, x, \hat{a}) &= x + 1 \\ \text{left}(a, x) &= x \\ h(X) &= \max X \end{aligned}$$

(c)

$N \rightarrow \text{nil}_{\epsilon} \mid C^c \mid S \quad \ldots h$

$S \rightarrow \bar{L} \mid \text{split}_{L^c\ C^c} \quad \ldots h$

$C \rightarrow \text{single}_{a} \mid \text{pair}_{a\ \mathrm{N}\ \hat{a}} \quad \ldots h$

$C^c \rightarrow C \setminus_h S$

$\bar{L} \rightarrow \text{left}_{\mathrm{a}\ N} \quad \ldots h$

$L^c \rightarrow N \setminus_h \bar{L}$

Figure 6.3: **ADP formulation of Nussinov's algorithm including sparsification.** Sparsified Nussinov grammar with (a) OCT and (c) OCT-STEP sparsification and (b) the corresponding grammar. The number of split points for S are significantly reduced by applying sparsification. C^c has a parse for subsequence $A_{i..j}$ only if $A_{i..j}$ fulfills the OCT criterion and L^c only if $A_{i..j}$ fulfills the STEP criterion. L^c can be derived by introducing a new nonterminal $\bar{L}$ that covers the case where the left position is unpaired. Figure adapted from [MSZ11].

For that, we introduce the new rule $C^c \rightarrow C \setminus_h S$ to sparsify C with our novel sparsification operator (see Definition 6.4). The modified grammar of the original ADP formulation of Nussinov's algorithm (Figure 6.2) is shown in Figure 6.3a and the corresponding algebra in Figure 6.3b. We show in the following that C^c represents the OCT fragments.

Proposition 6.1 *$C^c(i, j) = C(i, j)$ iff $A_{i..j}$ fulfills the OCT criterion for $h(X) = \max X$.*

Proof. As subsequences of length 1 cannot be split, S does not have a parse for those subsequences. Consequently, $C^c(i, i) = C(i, i)$ follows directly from Definition 6.4 for all i. So let us consider subsequences of at least length 2 and let $C(i, j) = \{c_{ij}\}$ and $S(i, j) = \{s_{ij}\}$.

1. $A_{i..j}$ fulfills the OCT criterion, i.e. $c_{ij} > s_{ij}$:
   $h(C(i, j) \uplus S(i, j)) = h(C(i, j)) = h(C(i, j) - C'(i, j)) \neq h(S(i, j))$
   implies $C'(i, j) = \emptyset$ and thus $C^c(i, j) = C(i, j)$.

2. $A_{i..j}$ does not fulfill the OCT criterion, i.e. $c_{ij} \leq s_{ij}$:
   $h(C(i,j) \uplus S(i,j)) = h(S(i,j)) = h((C(i,j) - C'(i,j)) \uplus S(i,j))$
   holds for $C'(i,j) = C(i,j)$ and thus $C^c(i,j)$ is empty. □

Thus, C^c has a parse for subsequence $A_{i..j}$ only if $A_{i..j}$ fulfills the OCT criterion and can be used in rule N and, more importantly, in S (see Figure 6.3a).

The runtime of the algorithm can be further reduced by additionally integrating the STEP criterion (cf. Definition 6.3). The grammar is displayed in Figure 6.3c and the corresponding algebra in Figure 6.3b. For the split point $q = i + 1$, we need to introduce a new rule $\bar{L}$ that covers the case where the left position is unpaired. The sparse table L^c (for left paired) represents fragments where the left position is paired and can be created from $\bar{L}$ by introducing the new rule $L^c \rightarrow N \setminus_h \bar{L}$. We show in the following that L^c represents the STEP fragments.

Proposition 6.2 *$L^c(i,j) = N(i,j)$ iff $A_{i..j}$ fulfills the STEP criterion for $h(X) = \max X$.*

Proof. Let $N(i,j) = \{n_{ij}\}$ and $\bar{L}(i,j) = \{\bar{l}_{ij}\}$.

1. $A_{i..j}$ fulfills the STEP criterion, i.e. $n_{ij} > n_{ii} + n_{i+1j} = \bar{l}_{ij}$:
   $h(N(i,j) \uplus \bar{L}(i,j)) = h(N(i,j)) = h(N(i,j) - N'(i,j)) \neq h(\bar{L}(i,j))$
   implies $N'(i,j) = \emptyset$ and thus $L^c(i,j) = N(i,j)$.
2. $A_{i..j}$ does not fulfills the STEP criterion, i.e. $n_{ij} \leq n_{ii} + n_{i+1j} = \bar{l}_{ij}$:
   $h(N(i,j) \uplus \bar{L}(i,j)) = h(\bar{L}(i,j)) = h((N(i,j) - N'(i,j)) \uplus \bar{L}(i,j))$
   holds for $N'(i,j) = N(i,j)$ and thus $L^c(i,j)$ is empty. □

Thus, L^c has a parse for subsequence $A_{i..j}$ only if $A_{i..j}$ fulfills the STEP criterion. We know that there exists an optimal split point q for $A_{i..j}$ such that either $q = i + 1$, or $A_{i..q-1}$ fulfills the STEP and $A_{q..j}$ the OCT criterion. Consequently, C^c and L^c can both be used in the grammar rule for nonterminal S when we also check for fragments that are unpaired on the left side (represented by $\bar{L}$), see Figure 6.3c. This leads to a reduction in split points that need to be considered during calculation.

6.3.2 Implementation

To implement the sparsification operator, the parser must be extended for grammar rules like $C^c \rightarrow C \setminus_h S$. Once table entries $S(i,j)$ and $C(i,j) = \{b_1, \ldots, b_k\}$ are computed, $C^c(i,j)$ is computed as follows. We initialize $C^c(i,j) = \emptyset$ and check sequentially for each b_x where $1 \leq x \leq k$ if

$$h(C^c(i,j) \uplus \{b_x, \ldots, b_k\} \uplus S(i,j)) = h(C^c(i,j) \uplus \{b_{x+1}, \ldots, b_k\} \uplus S(i,j)).$$

If this condition holds, b_x is discarded. If not, b_x is added to $C^c(i,j)$. This means that b_x can be discarded if it does not contribute to the solution when considering elements in $S(i,j)$ and all elements in $C(i,j)$ that have not been removed, i.e. $C^c(i,j) \uplus \{b_{x+1}, \ldots, b_k\}$. If we assume that the solution sets have constant length (usually even singleton sets), this check requires constant time.

The main benefit of sparsification happens when C^c occurs on the right hand side of some other grammar rule, e.g. the rule for S in Figure 6.3a. Usually, to compute $S(i,j)$, the parser would consider each of the linearly many possible split points q to combine $N(i,q-1)$ with $C^c(q,j)$. But if C^c is sparse, all q for which $C^c(q,j)$ is empty can be discarded. The possibility to go from one non-empty entry of C^c to the next one is hence essential to reduce the number of split points q to be considered. To ensure a faster runtime as well, the next non-empty entry has to be computed in constant time. To benefit from sparsification also in terms of space complexity, C^c should be stored in some kind of sparse table data structure (and C should not be stored at all). This can be achieved, for example, by representing the sparse table by lists containing the non-empty entries of each individual row (or column).

6.3.3 Advanced choice functions

So far, sparsification has only been applied to simple maximization and minimization problems. However, Definition 6.4 generalizes naturally to more complex scoring schemes. In the grammars of Figure 6.3 we can, for example, enumerate the k best suboptimal solutions by changing the choice function h to choose – instead of the maximal score – the multiset of scores of the k best solutions. In this case, by Definition 6.4, $C^c(i,j)$ contains exactly the scores of the *sub*optimally co-terminus structures in the sense that the structure is among the k best structures and contains the base pair (i,j). This means that we keep exactly those solutions that cannot be discarded based on the current available information. This keeps the tables as sparse as possible but still guarantees that we can reconstruct the k best solutions using standard back-tracing techniques.

6.4 Results

In order to evaluate the effectiveness of the sparsification approaches introduced in Section 6.2, we extended the existing Haskell implementation of ADP [GMS04] with our sparsification operator. Furthermore, we implemented the different Nussinov variants in order to measure their degree of sparseness. Note that the Haskell implementation of ADP is intended as a flexible and extensible platform for quick experiments and is not optimized for performance. Therefore, this section is only a feasibility evaluation for which we used the Haskell interpreter hugs

Table 6.1: **Number of split points for Nussinov variants.** Average number of split points over 100 random sequences (a) for finding the optimal solution and (b) for enumerating k suboptimal solutions for sequences of length 50. In the latter case, the number of split point instances is the number of combinations of suboptimal solutions at a split point. For both analyses, the average proportion of split points examined by the sparse variants compared with the original Nussinov algorithm is given in brackets. Table taken from [MSZ11].

(a)

Nussinov sparsification	Average number of split points per sequence	
	of length 100	of length 200
none (Figure 6.2)	65288	510473
OCT (Figure 6.3a)	10620 (16.3 %)	53947 (10.6 %)
OCT-STEP (Figure 6.3c)	9430 (14.4 %)	43679 (8.6 %)

(b)

Nussinov sparsification	Average number of split point instances per sequence		
	$k = 1$	$k = 5$	$k = 10$
none (Figure 6.2)	8573	128461	416778
OCT (Figure 6.3a)	2277 (26.6 %)	26032 (20.3 %)	77986 (18.7 %)
OCT-STEP (Figure 6.3c)	2124 (24.8 %)	17576 (13.7 %)	49595 (11.9%)

to measure the number of split points of the different Nussinov variants, Nussinov's algorithm without sparsification, with OCT sparsification and with OCT-STEP sparsification.

6.4.1 Sparsified variants of Nussinov's algorithm

We took 100 random sequences of length 100 and 200 and folded them using our implementation of the non-sparse Nussinov variant, the OCT variant and the OCT-STEP variant. The average number of split points that need to be examined during the recursive calculation is given in Table 6.1a. Also the average proportion of split points examined by the sparse variants compared with the original Nussinov algorithm are shown. One can see that the proportion of split points analyzed by the sparse variants compared with the original Nussinov algorithm decreases with increasing sequence length. For the OCT variant, 16.3% and 10.6% of the split points of the original Nussinov algorithm have to be examined for sequences of length 100 and 200, respectively. The OCT-STEP variant is slightly more sparse than the OCT variant. Note that we compare our sparse variants with the original Nussinov algorithm with a recursion $S \to NC$ (cf. Figure 6.2) that considers only split points where the outermost bases of C can pair. The basic Nussinov variant examines significantly more split points due to a recursion of

the form $S \to NN$. Compared with this basic variant, a recursion of the form $S \to NC$ reduces the number of split points by a constant factor independent of the sequence length. In contrast, the sparsification potential of the sparsified variants increases with increasing sequence length. Nevertheless, we also compared our implementation with the basic variant and could confirm the results reported in [BTZZU11].

6.4.2 Enumerating suboptimal solutions

In a second experiment, we evaluated the sparsification potential for advanced choice functions by enumerating the best k suboptimal solutions. To measure the effectiveness of sparsification in this scenario, we counted the number of split point instances examined during bottom-up parsing. At each split point, there can be several split point instances that represent the number of combinations of suboptimal solutions. When sparsification is applied, the number of suboptimal solutions that need to be stored can be reduced. We computed for 100 random sequences of length 50 the average number of split points examined during the evaluation of suboptimal solutions. The results are shown in Table 6.1b. We also show the average proportion of split points examined by the sparse variants compared with the original Nussinov algorithm. For the OCT variant, we achieve a reduction of split points to 26.6%, 20.3% and 18.7% when we enumerate 1, 5 and 10 suboptimal solutions, respectively. Consistent with the results for computing the optimal solution, the OCT-STEP variant is slightly more sparse than the OCT variant. In summary, the proportion of split points analyzed by the sparse variants compared with the original Nussinov algorithm decreases with increasing number of suboptimal solutions k.

6.5 Discussion

We presented the first systematic approach to describe sparsification in dynamic programming in terms of a general framework. For that, we defined a universal sparsification operator, which we integrated into the ADP framework. We described on the basis of the OCT and STEP criteria for the Nussinov algorithm how it can be used to easily incorporate sparsification criteria into ADP programs. In a feasibility study[3], we showed that the number of split points that have to be examined during the evaluation of the DP recursions are significantly reduced for the sparsified Nussinov variants. We demonstrated that this behavior extends also to more advanced choice functions, as the enumeration of suboptimal solutions. Importantly, the potential for speedup increases when the size of the search space grows, for example when the sequence length or the number of suboptimal solutions increases. Our novel sparsifica-

[3]For implementations of practical relevance, an analogous extension of the ADP to C++ compiler Bellman's GAP [SJG11] is required.

tion operator can be used analogously for more complex algorithms for RNA structure prediction [WZZU07], simultaneous alignment and folding [ZUGVWS08], RNA-RNA-interaction prediction [SMW+10], and the prediction of RNA pseudoknot structures [MSW+10].

CHAPTER 7

Conclusion

In this thesis, we introduced novel strategies for the functional characterization of ncRNAs, including the novel ensemble-based sparsification, a fast algorithm for detecting sequence-structure motifs shared by two RNAs and two fast methods for simultaneous alignment and folding. On top of that, we introduced a novel framework to combine the concepts of sparsification and dynamic programming (DP).

The essential basis for the subsequent novel approaches is described in Chapter 3. We demonstrated that our novel ensemble-based sparsification has the potential to speed up various RNA analysis methods without compromising the quality. The main idea is to restrict the search space by removing subsolutions that are unlikely in the structure ensembles of the RNA sequences. We demonstrated its practicability for speeding up the computation of sequence-structure patterns between two RNAs (Chapter 4) and simultaneous alignment and folding (Chapter 5). Since ensemble-based sparsification is a universal method, it can be applied to other RNA-related tasks as well and therefore allows for future advancements of other algorithms. One possible application area is to speed up the tool LocARNA-P [WJH+12], which computes RNA alignment reliabilities from simultaneous alignment and folding partition functions.

In Chapter 4, we presented the tool ExpaRNA-P, a novel algorithm for simultaneous pattern matching and folding. The main goal is to identify local patterns shared by two RNAs on the basis of sequence *and* structure information. Previous approaches, such as the "predecessor" ExpaRNA [HWBB09], require a fixed structure for each input sequence that can be predicted only unreliably in many cases. For the first time, the pattern matching is realized in the entire Boltzmann-distributed structure ensembles. Our novel ensemble-based sparsification (cf. Chapter 3) identifies those structural parts of the ensembles that are likely to occur and thus drastically reduces the computational complexity. We demonstrated that each position is con-

sidered in only a constant number of base pairs. For this reason, ExpaRNA-P runs in $O(n^2)$ time, which is as efficient as plain sequence alignment. To evaluate the usefulness of our novel algorithm, we devised the pipeline ExpLoc-P, where the patterns identified by ExpaRNA-P are utilized to speed up the tool LocARNA for simultaneous alignment and folding [WRH$^+$07]. The patterns are first chained to find the best subset that can be part of an alignment simultaneously. These patterns are subsequently used as anchor constraints to guide the alignment. As only the parts in between those anchors need to be computed with the more time-consuming tool LocARNA, a significant speedup can be achieved compared with using LocARNA alone. To show that these considerations also work in practice, we ran extensive benchmarks on the well-established BRAliBase 2.1 k2 benchmark set [WMS06, GWW05] where we compared our novel tool ExpaRNA-P with ExpaRNA and the widely used tools LocARNA and RAF [DFB08]. RAF is currently the fastest simultaneous alignment and folding tool due to its heuristic filtering based on sequence alignments. We demonstrated that ExpaRNA-P offers the best compromise between speed and quality, where the quality was comparable with LocARNA's and still a considerable speedup of about four-fold was achieved compared with LocARNA. RAF on the other hand cannot keep up the alignment quality in the benchmark for low sequence identities that are particularly difficult to align. This is especially important since they are usually considered the most decisive RNA alignment instances. Thus, ExpaRNA-P is the only tool that maintains a high alignment quality for all sequence identity ranges and achieves a significant speedup. Furthermore, we showed that for long sequences of around 400nt from the BRAliBase 2.1 benchmark set, the speedup is more than 30-fold. This demonstrates an increase in the effectiveness of our novel approach when the RNA sequences get longer, which is crucial for large scale analysis. Moreover, we formally introduced and evaluated strict and relaxed variants of our novel algorithm; the latter makes the approach sensitive to compensatory mutations. The advantage of the relaxed variant was confirmed by an increased coverage in the low sequence identity region compared with the strict variant. Furthermore, we suggested in addition to the suboptimal traceback a heuristic traceback that reduces the number of enumerated sequence-structure patterns. All these different ExpLoc-P variants enable tailoring of the method to specific application scenarios.

In Chapter 5, we introduced the two novel algorithms PARSE and SPARSE for simultaneous alignment and folding. Importantly, PARSE combines for the first time the original structure prediction flexibility of Sankoff's algorithm [San85] with PMcomp's lightweight energy model [HBS04]. The lightweight energy model scores structural features based on base pair probabilities that are derived from a full-featured energy model by McCaskill's algorithm [McC90]. Going far beyond PMcomp's potential, PARSE allows loop deletions and insertions such that stems of different length can be aligned. Thus, analogous to Sankoff's algorithm, PARSE computes two potentially different but compatible structures for the two RNA sequences. By filtering the base pairs according to their probabilities by a constant threshold

as implemented in LocARNA [WRH+07], we obtain LocARNA's $O(n^4)$ time and $O(n^2)$ space complexity. While this constitutes already a relevant contribution, we additionally built upon PARSE's model to derive the sparsified variant SPARSE that integrates the novel ensemble-based sparsification (cf. Chapter 3). Thus, SPARSE enables for the first time simultaneous alignment and folding in quadratic time without employing sequence based heuristics. This advantage is demonstrated by benchmarks on BRAliBase 2.1 benchmark sets where we compare our novel tool SPARSE with LocARNA and RAF [DFB08]. Whereas SPARSE and RAF achieve similar speedups of around 4 over LocARNA on benchmark set k2, SPARSE – contrary to RAF – keeps up the alignment quality even for 'hard' alignment instances with low sequence identity. This superiority of SPARSE over RAF is also confirmed for multiple alignment on the basis of three-way alignments in BRAliBase 2.1. Finally, to demonstrate the effectiveness of loop insertions and deletions, we compared SPARSE's structure prediction ability to LocARNA's. Measured by Matthews correlation coefficient (MCC), we showed an increased structure prediction quality for SPARSE compared with LocARNA for all sequence identity ranges. To exemplify the importance of a flexible structure prediction model, we showed a concrete example where SPARSE's ability to match different-sized stems leads to overall more stable structures and a higher structure prediction quality.

In Chapter 6, we introduced a framework for combining the two important concepts of algebraic dynamic programming (ADP) and sparsification. Unlike our novel ensemble-based sparsification technique introduced in Chapter 3 that discards unlikely subsolutions, the sparsification approach introduced in [WZZU07] does not sacrifice optimality. Sparsification was used in various RNA-related tasks and has the potential to speed up dynamic programming computations. ADP is a formalism that allows defining DP algorithms in a simple and formally precise way [GMS04]. Since there is an explicit separation of the recursive structure and the evaluation, ADP programs can be easily adapted to different scoring schemes. To incorporate sparsification into the ADP framework, we proposed a novel sparsification operator that can be easily applied within the ADP recursions. Thereby, it was for the first time possible to study these two relevant techniques within a common framework. We demonstrated the simplicity of use and effectiveness of the novel sparsification operator on the basis of Nussinov's algorithm for RNA structure prediction [NPGK78]. The number of split points – that are ultimately decisive for the algorithm's runtime – were significantly reduced. Crucially, the speedup potential enlarges with increasing complexity of the problem, which we showed for the length of an RNA molecule and the number of enumerated suboptimal solutions.

To sum up, this thesis covers novel concepts and approaches for the functional classification of ncRNAs. In particular, we focus on designing fast algorithms which, nevertheless, guarantee high quality output. We demonstrated the superior performance of our novel approaches compared with state-of-the-art programs on real RNA sequences. Furthermore, our novel ensemble-based sparsification technique has the potential to speed up other algorithms for

RNA analysis. Both forms of sparsification studied in this thesis are valuable techniques to make DP algorithms more efficient in the future. In this way, we make important contributions to a better understanding of the functionalities of ncRNA molecules.

Bibliography

[Aku99] Tatsuya Akutsu. Approximation and exact algorithms for RNA secondary structure prediction and recognition of stochastic context-free languages. *Journal of Combinatorial Optimization*, 3:321–336, 1999.

[BBB+08] Athanasius F. Bompfünewerer, Rolf Backofen, Stephan H. Bernhart, Jana Hertel, Ivo L. Hofacker, Peter F. Stadler, and Sebastian Will. Variations on RNA folding and alignment: lessons from Benasque. *Journal of Mathematical Biology*, 56(1-2):129–144, 2008.

[BCA14] Laetitia Bourgeade, Cedric Chauve, and Julien Allali. Chaining sequence/structure seeds for computing RNA similarity. In *Proceedings of 1st workshop on Computational Methods for Structural RNAs (CMSR'14)*, pages 1–12. McGill University, 2014.

[BDE+13] Sarah W. Burge, Jennifer Daub, Ruth Eberhardt, John Tate, Lars Barquist, Eric P. Nawrocki, Sean R. Eddy, Paul P. Gardner, and Alex Bateman. Rfam 11.0: 10 years of RNA families. *Nucleic Acids Res*, 41(Database issue):D226–32, 2013.

[Bel57] Richard Bellman. *Dynamic Programming*. Princeton University Press, Princeton, NJ, USA, 1 edition, 1957.

[BHW+08] Stephan H. Bernhart, Ivo L. Hofacker, Sebastian Will, Andreas R. Gruber, and Peter F. Stadler. RNAalifold: improved consensus structure prediction for RNA alignments. *BMC Bioinformatics*, 9:474, 2008.

[BS07] Rolf Backofen and Sven Siebert. Fast detection of common sequence structure patterns in RNAs. *Journal of Discrete Algorithms*, 5(2):212–228, 2007.

[BTZZU11] Rolf Backofen, Dekel Tsur, Shay Zakov, and Michal Ziv-Ukelson. Sparse RNA folding: Time and space efficient algorithms. *J. Discrete Algorithms*, 9(1):12–31, 2011.

[BW04] Rolf Backofen and Sebastian Will. Local sequence-structure motifs in RNA.

Journal of Bioinformatics and Computational Biology (JBCB), 2(4):681–698, 2004.

[CAS+11] Michael B. Clark, Paulo P. Amaral, Felix J. Schlesinger, Marcel E. Dinger, Ryan J. Taft, John L. Rinn, Chris P. Ponting, Peter F. Stadler, Kevin V. Morris, Antonin Morillon, Joel S. Rozowsky, Mark B. Gerstein, Claes Wahlestedt, Yoshihide Hayashizaki, Piero Carninci, Thomas R. Gingeras, and John S. Mattick. The reality of pervasive transcription. *PLoS Biol*, 9(7):e1000625; discussion e1001102, 2011.

[CCKT83] J. M. Chambers, W. S. Cleveland, Beat Kleiner, and Paul A. Tukey. *Graphical Methods for Data Analysis*. Wadsworth, 1983.

[Cle81] W. S. Cleveland. Lowess: A program for smoothing scatterplots by robust locally weighted regression. *The American Statistician*, 35(54), 1981.

[Con07] 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.

[Cri66] FHC Crick. Codon—anticodon pairing: the wobble hypothesis. *Journal of molecular biology*, 19(2):548–555, 1966.

[CS14] Thomas R Cech and Joan A Steitz. The noncoding RNA revolution—trashing old rules to forge new ones. *Cell*, 157(1):77–94, 2014.

[DE06] R. D. Dowell and S. R. Eddy. Efficient pairwise RNA structure prediction and alignment using sequence alignment constraints. *BMC Bioinformatics*, 7:400, 2006.

[DFB08] Chuong B. Do, Chuan-Sheng Foo, and Serafim Batzoglou. A max-margin model for efficient simultaneous alignment and folding of RNA sequences. *Bioinformatics*, 24(13):i68–76, 2008.

[DGB06] Chuong B Do, Samuel S Gross, and Serafim Batzoglou. CONTRAlign: discriminative training for protein sequence alignment. In *Research in Computational Molecular Biology*, pages 160–174. Springer, 2006.

[DWB06] Chuong B Do, Daniel A Woods, and Serafim Batzoglou. CONTRAfold: RNA secondary structure prediction without physics-based models. *Bioinformatics*, 22(14):e90–e98, 2006.

[EB06] Robert C Edgar and Serafim Batzoglou. Multiple sequence alignment. *Current opinion in structural biology*, 16(3):368–373, 2006.

[Edg04] Robert C. Edgar. MUSCLE: multiple sequence alignment with high accuracy and high throughput. *Nucleic Acids Res*, 32(5):1792–7, 2004.

[EGGI92] David Eppstein, Zvi Galil, Raffaele Giancarlo, and Giuseppe F. Italiano. Sparse dynamic programming I: linear cost functions. *J. ACM*, 39(3):519–545, 1992.

[Est11] Manel Esteller. Non-coding RNAs in human disease. *Nat Rev Genet*, 12(12):861–74, 2011.

[FG10] Yelena Frid and Dan Gusfield. A simple, practical and complete O(n3/log n)-time algorithm for RNA folding using the Four-Russians speedup. *Algorithms Mol Biol*, 5:13, 2010.

[FPM05] Martin C Frith, Michael Pheasant, and John S Mattick. Genomics: The amazing complexity of the human transcriptome. *European Journal of Human Genetics*, 13(8):894–897, 2005.

[FXM+98] Andrew Fire, SiQun Xu, Mary K Montgomery, Steven A Kostas, Samuel E Driver, and Craig C Mello. Potent and specific genetic interference by double-stranded RNA in caenorhabditis elegans. *Nature*, 391(6669):806–811, 1998.

[GG04] Paul P. Gardner and Robert Giegerich. A comprehensive comparison of comparative RNA structure prediction approaches. *BMC Bioinformatics*, 5:140, 2004.

[GHR80] S. L. Graham, M. A. Harrison, and W. L. Ruzzo. An improved context-free recognizer. *ACM Transactions on Programming Languages and Systems*, 2(3):415–462, 1980.

[GHS97] J. Gorodkin, L. J. Heyer, and G. D. Stormo. Finding common sequence and structure motifs in a set of RNA sequences. In *Proc. of the 5th Int. Conf. on Intelligent Systems for Molecular Biology (ISMB'97)*, volume 5, pages 120–3, 1997.

[GM07] Ilan Gronau and Shlomo Moran. Optimal implementations of UPGMA and other common clustering algorithms. *Information Processing Letters*, 104(6):205–210, 2007.

[GMS04] Robert Giegerich, Carsten Meyer, and Peter Steffen. A discipline of dynamic programming over sequence data. *Sci. Comput. Program.*, 51(3):215–263, 2004.

[Got82] O. Gotoh. An improved algorithm for matching biological sequences. *J Mol Biol*, 162:705–708, 1982.

[GS06] Robert Giegerich and Peter Steffen. Challenges in the compilation of a domain specific language for dynamic programming. In *Proceedings of the 2006 ACM symposium on Applied computing*, SAC '06, pages 1603–1609, New York, NY, USA, 2006. ACM.

[GWW05] Paul P. Gardner, Andreas Wilm, and Stefan Washietl. A benchmark of mul-

tiple sequence alignment programs upon structural RNAs. *Nucleic Acids Res*, 33(8):2433–9, 2005.

[HBS04] I. L. Hofacker, S. H. Bernhart, and P. F. Stadler. Alignment of RNA base pairing probability matrices. *Bioinformatics*, 20(14):2222–7, 2004.

[HFS+94] Ivo L. Hofacker, Walter Fontana, Peter F. Stadler, Sebastian Bonhoeffer, Manfred Tacker, and Peter Schuster. Fast folding and comparison of RNA secondary structures. *Monatshefte Chemie*, 125:167–188, 1994.

[HFS02] Ivo L. Hofacker, Martin Fekete, and Peter F. Stadler. Secondary structure prediction for aligned RNA sequences. *J Mol Biol*, 319(5):1059–66, 2002.

[HLSG05] Jakob Hull Havgaard, Rune B. Lyngso, Gary D. Stormo, and Jan Gorodkin. Pairwise local structural alignment of RNA sequences with sequence similarity less than 40%. *Bioinformatics*, 21(9):1815–24, 2005.

[Hol05] Ian Holmes. Accelerated probabilistic inference of RNA structure evolution. *BMC Bioinformatics*, 6:73, 2005.

[HSM07] Arif Ozgun Harmanci, Gaurav Sharma, and David H. Mathews. Efficient pairwise RNA structure prediction using probabilistic alignment constraints in Dynalign. *BMC Bioinformatics*, 8:130, 2007.

[HTG07] Jakob H. Havgaard, Elfar Torarinsson, and Jan Gorodkin. Fast pairwise structural RNA alignments by pruning of the dynamical programming matrix. *PLoS Comput Biol*, 3(10):1896–908, 2007.

[HTGK03] Matthias Höchsmann, Thomas Töller, Robert Giegerich, and Stefan Kurtz. Local similarity in RNA secondary structures. In *Proceedings of Computational Systems Bioinformatics (CSB 2003)*, volume 2, pages 159–168. IEEE Computer Society, 2003.

[HVG04] Matthias Höchsmann, Björn Voß, and Robert Giegerich. Pure multiple RNA secondary structure alignments: a progressive profile approach. *IEEE/ACM Trans Comput Biol Bioinform*, 1(1):53–62, 2004.

[HWBB09] Steffen Heyne, Sebastian Will, Michael Beckstette, and Rolf Backofen. Lightweight comparison of RNAs based on exact sequence-structure matches. *Bioinformatics*, 25(16):2095–2102, 2009.

[HzS12] Christian Höner zu Siederdissen. Sneaking around concatMap: efficient combinators for dynamic programming. In *ACM SIGPLAN Notices*, volume 47, pages 215–226. ACM, 2012.

[JLMZ02] Tao Jiang, Guohui Lin, Bin Ma, and Kaizhong Zhang. A general edit distance between RNA structures. *J Comput Biol*, 9(2):371–88, 2002.

[Jus01] Winfried Just. Computational complexity of multiple sequence alignment with SP-score. *Journal of Computational Biology*, 8(6):615–623, 2001.

[JWZ95] T. Jiang, J. Wang, and K. Zhang. Alignment of trees - an alternative to tree edit. *Theoretical Computer Science*, 143(1):137–148, 1995.

[KH99] B Knudsen and J Hein. RNA secondary structure prediction using stochastic context-free grammars and evolutionary history. *Bioinformatics*, 15(6):446–54, 1999.

[KH03] Bjarne Knudsen and Jotun Hein. Pfold: RNA secondary structure prediction using stochastic context-free grammars. *Nucleic Acids Res*, 31(13):3423–8, 2003.

[KMKM02] Kazutaka Katoh, Kazuharu Misawa, Kei-ichi Kuma, and Takashi Miyata. MAFFT: a novel method for rapid multiple sequence alignment based on fast Fourier transform. *Nucleic Acids Res*, 30(14):3059–66, 2002.

[KR08] Daniel H Kim and John J Rossi. RNAi mechanisms and applications. *Biotechniques*, 44(5):613, 2008.

[LBHZS+11] Ronny Lorenz, Stephan H. Bernhart, Christian Höner Zu Siederdissen, Hakim Tafer, Christoph Flamm, Peter F. Stadler, and Ivo L. Hofacker. ViennaRNA Package 2.0. *Algorithms Mol Biol*, 6:26, 2011.

[LMWZU09] Y. Lifshits, S. Mozes, O. Weimann, and M. Ziv-Ukelson. Speeding up HMM decoding and training by exploiting sequence repetitions. *Algorithmica*, 54(3):379–399, 2009.

[LS10] Christian Laing and Tamar Schlick. Computational approaches to 3D modeling of RNA. *Journal of Physics: Condensed Matter*, 22(28):283101, 2010.

[LS11] Christian Laing and Tamar Schlick. Computational approaches to RNA structure prediction, analysis, and design. *Current opinion in structural biology*, 21(3):306–318, 2011.

[Mat75] B. W. Matthews. Comparison of the predicted and observed secondary structure of T4 phage lysozyme. *Biochim Biophys Acta*, 405(2):442–51, 1975.

[McC90] J. S. McCaskill. The equilibrium partition function and base pair binding probabilities for RNA secondary structure. *Biopolymers*, 29(6-7):1105–19, 1990.

[MM06] John S Mattick and Igor V Makunin. Non-coding RNA. *Human molecular genetics*, 15(suppl 1):R17–R29, 2006.

[MSW+10] Mathias Möhl, Raheleh Salari, Sebastian Will, Rolf Backofen, and S. Cenk Sahinalp. Sparsification of RNA structure prediction including pseudoknots. *Algorithms Mol Biol*, 5(1):39, 2010.

[MSZ11] Mathias Möhl, Christina Schmiedl, and Shay Zakov. Sparsification in algebraic

dynamic programming. In *Proceedings of the German Conference on Bioinformatics (GCB 2011)*, 2011.

[MSZT99] DH Mathews, J Sabina, M Zuker, and DH Turner. Expanded sequence dependence of thermodynamic parameters improves prediction of RNA secondary structure. *J Mol Biol*, 288(5):911–40, 1999.

[MT02] David H. Mathews and Douglas H. Turner. Dynalign: an algorithm for finding the secondary structure common to two RNA sequences. *J Mol Biol*, 317(2):191–203, 2002.

[MTF10] John S. Mattick, Ryan J. Taft, and Geoffrey J. Faulkner. A global view of genomic information–moving beyond the gene and the master regulator. *Trends in Genetics*, 26(1):21–8, 2010.

[NHH00] C. Notredame, D. G. Higgins, and J. Heringa. T-Coffee: A novel method for fast and accurate multiple sequence alignment. *J Mol Biol*, 302(1):205–17, 2000.

[NPGK78] Ruth Nussinov, George Pieczenik, Jerrold R. Griggs, and Daniel J. Kleitman. Algorithms for loop matchings. *SIAM J Appl Math*, 35(1):68–82, July 1978.

[NW70] S. B. Needleman and C. D. Wunsch. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *J Mol Biol*, 48(3):443–53, 1970.

[Ohn72] Susumu Ohno. So much "junk" DNA in our genome. In *Brookhaven Symp Biol*, volume 23, pages 366–370, 1972.

[OMH+14] Christina Otto, Mathias Möhl, Steffen Heyne, Mika Amit, Gad M. Landau, Rolf Backofen, and Sebastian Will. ExpaRNA-P: simultaneous exact pattern matching and folding of RNAs. *BMC Bioinformatics*, 15(1):404, 2014.

[OWB08] 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.

[PBS+06] J. S. Pedersen, G. Bejerano, A. Siepel, K. Rosenbloom, K. Lindblad-Toh, E. S. Lander, J. Kent, W. Miller, and D. Haussler. Identification and Classification of Conserved RNA Secondary Structures in the Human Genome. *PLoS Comput Biol*, 2(4):e33, 2006.

[RBT+10] Mathieu Rederstorff, Stephan H. Bernhart, Andrea Tanzer, Marek Zywicki, Katrin Perfler, Melanie Lukasser, Ivo L. Hofacker, and Alexander Huttenhofer. RNPomics: defining the ncRNA transcriptome by cDNA library generation from ribonucleo-protein particles. *Nucleic Acids Res*, 38(10):e113, 2010.

[RSG07] Jens Reeder, Peter Steffen, and Robert Giegerich. pknotsRG: RNA pseudoknot folding including near-optimal structures and sliding windows. *Nucleic Acids Res*, 35(Web Server issue):W320–4, 2007.

[RSHG04] Marc Rehmsmeier, Peter Steffen, Matthias Höchsmann, and Robert Giegerich. Fast and effective prediction of microRNA/target duplexes. *RNA*, 10(10):1507–17, 2004.

[RSZ04] Jianhua Ruan, Gary D. Stormo, and Weixiong Zhang. An iterated loop matching approach to the prediction of RNA secondary structures with pseudoknots. *Bioinformatics*, 20(1):58–66, 2004.

[San72] David Sankoff. Matching sequences under deletion/insertion constraints. *Proceedings of the National Academy of Sciences*, 69(1):4–6, 1972.

[San85] David Sankoff. Simultaneous solution of the RNA folding, alignment and protosequence problems. *SIAM J. Appl. Math.*, 45(5):810–825, 1985.

[SB05] 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.

[Sel74] P.H. Sellers. On the theory and computation of evolutionary distances. *SIAM J. Appl. Math*, 26:787–793, 1974.

[SGSM13] Martin A. Smith, Tanja Gesell, Peter F. Stadler, and John S. Mattick. Widespread purifying selection on RNA structure in mammals. *Nucleic Acids Res*, 2013.

[SJG11] Georg Sauthoff, Stefan Janssen, and Robert Giegerich. Bellman's GAP - A Declarative Language for Dynamic Programming. In *13th International ACM SIGPLAN Symposium on Principles and Practice of Declarative Programming*, PPDP 2011. ACM, 2011.

[SMH+12] Christina Schmiedl, Mathias Möhl, Steffen Heyne, Mika Amit, Gad M. Landau, Sebastian Will, and Rolf Backofen. Exact pattern matching for RNA structure ensembles. In *Proceedings of the 16th International Conference on Research in Computational Molecular Biology (RECOMB 2012)*, volume 7262 of *Lecture Notes in Computer Science*, pages 245–260. Springer Berlin Heidelberg, 2012.

[SMW+10] Raheleh Salari, Mathias Möhl, Sebastian Will, S. Cenk Sahinalp, and Rolf Backofen. Time and space efficient RNA-RNA interaction prediction via sparse folding. In Bonnie Berger, editor, *Proc. of RECOMB 2010*, volume 6044 of *Lecture Notes in Computer Science*, pages 473–490. Springer-Verlag Berlin Heidelberg, 2010.

[SVR+06] Peter Steffen, Björn Voß, Marc Rehmsmeier, Jens Reeder, and Robert

Giegerich. RNAshapes: an integrated RNA analysis package based on abstract shapes. *Bioinformatics*, 22(4):500–3, 2006.

[SYKB07] Bruce A Shapiro, Yaroslava G Yingling, Wojciech Kasprzak, and Eckart Bindewald. Bridging the gap in RNA structure prediction. *Current opinion in structural biology*, 17(2):157–165, 2007.

[Tai79] K.-C. Tai. The tree-to-tree correction problem. *In Journal of the ACM*, 26(3):422–433, 1979.

[TB99] I. Jr Tinoco and C. Bustamante. How RNA folds. *J Mol Biol*, 293(2):271–81, 1999.

[The12] The ENCODE Project Consortium. An integrated encyclopedia of DNA elements in the human genome. *Nature*, 489(7414):57–74, 2012.

[THG94] J. D. Thompson, D. G. Higgins, and T. J. Gibson. CLUSTAL W: improving the sensitivity of progressive multiple sequence alignment through sequence weighting, position-specific gap penalties and weight matrix choice. *Nucleic Acids Res*, 22(22):4673–80, 1994.

[THG07] Elfar Torarinsson, Jakob H. Havgaard, and Jan Gorodkin. Multiple structural alignment and clustering of RNA sequences. *Bioinformatics*, 23(8):926–32, 2007.

[TM10] Douglas H. Turner and David H. Mathews. NNDB: the nearest neighbor parameter database for predicting stability of nucleic acid secondary structure. *Nucleic Acids Res*, 38(Database issue):D280–2, 2010.

[TSH+06] Elfar Torarinsson, Milena Sawera, Jakob H. Havgaard, Merete Fredholm, and Jan Gorodkin. Thousands of corresponding human and mouse genomic regions unalignable in primary sequence contain common RNA structure. *Genome Res*, 16(7):885–9, 2006.

[Val75] L.G. Valiant. General context-free recognition in less than cubic time. *Journal of Computer and System Sciences*, 10:308–315, 1975.

[WC53] J. D. Watson and F. H. Crick. Molecular structure of nucleic acids; a structure for deoxyribose nucleic acid. *Nature*, 171(4356):737–8, 1953.

[WHS05] Stefan Washietl, Ivo L. Hofacker, and Peter F. Stadler. Fast and reliable prediction of noncoding RNAs. *Proc Natl Acad Sci USA*, 102(7):2454–9, 2005.

[WJH+12] Sebastian Will, Tejal Joshi, Ivo L. Hofacker, Peter F. Stadler, and Rolf Backofen. LocARNA-P: Accurate boundary prediction and improved detection of structural RNAs. *RNA*, 18(5):900–14, 2012.

[WKS+11] Yue Wan, Michael Kertesz, Robert C. Spitale, Eran Segal, and Howard Y. Chang. Understanding the transcriptome through RNA structure. *Nat Rev Genet*, 12(9):641–55, 2011.

[WMS06] Andreas Wilm, Indra Mainz, and Gerhard Steger. An enhanced RNA alignment benchmark for sequence alignment programs. *Algorithms Mol Biol*, 1:19, 2006.

[WOM+15] Sebastian Will, Christina Otto, Milad Miladi, Mathias Möhl, and Rolf Backofen. SPARSE: Quadratic time simultaneous alignment and folding of RNAs without sequence-based heuristics. *Bioinformatics*, doi:10.1093/bioinformatics/btv185, first published online April 2, 2015.

[WRH+07] 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 Comput Biol*, 3(4):e65, 2007.

[WSM+13] Sebastian Will, Christina Schmiedl, Milad Miladi, Mathias Möhl, and Rolf Backofen. SPARSE: Quadratic time simultaneous alignment and folding of RNAs without sequence-based heuristics. In *Proceedings of the 17th International Conference on Research in Computational Molecular Biology (RECOMB 2013)*, volume 7821 of *Lecture Notes in Computer Science*, pages 289–290. Springer Berlin Heidelberg, 2013.

[WWH+12] Stefan Washietl, Sebastian Will, David A. Hendrix, Loyal A. Goff, John L. Rinn, Bonnie Berger, and Manolis Kellis. Computational analysis of noncoding RNAs. *Wiley Interdiscip Rev RNA*, 3(6):759–78, 2012.

[WYB13] Sebastian Will, Michael Yu, and Bonnie Berger. Structure-based whole-genome realignment reveals many novel noncoding RNAs. *Genome Res*, 23(6):1018–1027, 2013.

[WZZU07] Ydo Wexler, Chaya Zilberstein, and Michal Ziv-Ukelson. A study of accessible motifs and RNA folding complexity. *J Comput Biol*, 14(6):856–72, 2007.

[ZS81] M. Zuker and P. Stiegler. Optimal computer folding of large RNA sequences using thermodynamics and auxiliary information. *Nucleic Acids Res*, 9(1):133–48, 1981.

[ZS84] Michael Zuker and David Sankoff. RNA secondary structures and their prediction. *Bulletin of Mathematical Biology*, 46(4):591–621, 1984.

[ZTZU10] Shay Zakov, Dekel Tsur, and Michal Ziv-Ukelson. Reducing the worst case running times of a family of RNA and CFG problems, using Valiant's approach. In Vincent Moulton and Mona Singh, editors, *Proc. of the 10th Workshop on Algorithms in Bioinformatics (WABI)*, volume 6293 of *Lecture Notes in Computer Science*, pages 65–77. Springer Berlin / Heidelberg, 2010.

[ZUGVWS08] 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,

WABI 2008, volume 5251 of *Lecture Notes in Computer Science*, pages 174–185. Springer, 2008.

[ZUGVWS10] Michal Ziv-Ukelson, Irit Gat-Viks, Ydo Wexler, and Ron Shamir. A faster algorithm for simultaneous alignment and folding of RNA. *Journal of Computational Biology*, 17(8):1051–1065, 2010.

Abbreviations

A	adenine
ADP	algebraic dynamic programming
C	cytosine
CFG	context free grammar
CYK	Cocke-Younger-Kasami algorithm
DNA	deoxyribonucleic acid
DP	dynamic programming
EPM	exact pattern matching
G	guanine
HMM	hidden markov model
lncRNA	long ncRNA
MCC	Matthews correlation coefficient
mfe	minimum free energy
miRNA	microRNA
mRNA	messenger RNA
ncRNA	non-coding RNA
NMR	nuclear magnetic resonance
OCT	optimally co-terminus
pair SCFG	pairwise SCFG
RNA	ribonucleic acid
RNAi	RNA interference
rRNA	ribosomal RNA
SCFG	stochastic context free grammar

siRNA	small interfering RNA
snoRNA	small nucleolar RNA
snRNA	small nuclear RNA
tRNA	transfer RNA
U	uracil

List of mathematical expressions and symbols

$E_{SBI}(i,j,i',j')$	energy contribution of a 2-loop with closing base pair (i,j) and interior base pair (i',j') 13
μ	$\mu = \frac{1}{k_B \mathcal{T}}$, where $\mathcal{T}$ is the temperature and k_B is a constant 14
Z	partition function of the full sequence in the McCaskill algorithm 14
$Q(i,j)$	partition function for $A_{i..j}$ in the McCaskill algorithm 15
$Q^b(i,j)$	partition function for $A_{i..j}$, i and j are paired in the McCaskill algorithm 15
$Q^m(i,j)$	partition function for $A_{i..j}$ for parts of a multiloop for a non-empty structure in the McCaskill algorithm 15
$Q^{m1}(i,j)$	partition function for $A_{i..j}$ for parts of a multiloop with i left end of the single interior base pair in the McCaskill algorithm 15
$\Pr[(i,j)\|A]$	probability of base pair (i,j) in sequence A 16
P^A	set of all possible base pairs for sequence A 16
B	RNA sequence 17
$\|B\| = m$	length of B (with $m \leq n$) 17
T	secondary structure for B 17
$\mathcal{A}$	sequence alignment between sequences A and B 18
$\text{score}(\mathcal{A})$	score of $\mathcal{A}$ in sequence alignment 18
$\sigma(i,k)$	similarity between A_i and B_k in sequence alignment 18
$n_{\text{indel}}^{\mathcal{A}}$	number of gaps in $\mathcal{A}$ in sequence alignment 18
γ	uniform cost for each position in the gap in sequence alignment 18
$K(i,k)$	maximal similarity between $A_{1..i}$ and $B_{1..k}$ in sequence alignment 18
β	gap opening cost for affine gap costs 18
$(\mathcal{A},R,T)$	sequence-structure alignment; $\mathcal{A}$ sequence alignment, R structure for A, T structure for B 19
$\Pr[k \in \text{loop}(i,j)\|A]$	joint probability that a structure of A contains the base pair (i,j) and the unpaired base k such that (i,j) is the parent of k 27

$\Pr[(i',j') \in \text{loop}(i,j) \mid A]$	joint probability that a structure of A contains the base pairs (i,j) and (i',j') such that (i,j) is the parent of (i',j') 27
θ_1	threshold for base pair probabilities 28
$Q^{m2}(i,j)$	partition function for $A_{i..j}$ for parts of a multiloop with at least two interior base pairs 29
$\left\lvert \mathsf{pos}^A_{(i,j)} \right\rvert$	number of candidate positions in the loop closed by (i,j) in sequence A 33
$\mathsf{pos}^A_{(i,j)}(\bar{x})$	$\bar{x}$-th candidate position in the loop closed by (i,j) in sequence A 33
$\mathsf{mat\text{-}idx\text{-}bef}^A_{(i,j)}(i')$	first matrix index that lies before the sequence position i' in the loop closed by (i,j) in sequence A 33
$\text{mat-pos-bef}^{ijkl}(i',k')$	first matrix position that lies before sequence positions i' and k' 33
$(i \sim k)$	match in a pattern matching in ExpaRNA-P 38
$(ij \sim kl)$	base pair match in a pattern matching in ExpaRNA-P 38
$\mathcal{P} = (\mathcal{M}, \mathcal{S})$	pattern matching in ExpaRNA-P 38
$\mathcal{M}$	set of matches $(i \sim k)$ in a pattern matching in ExpaRNA-P 38
$\mathcal{S}$	set of base pair matches $(ij \sim kl)$ in a pattern matching in ExpaRNA-P 38
$\mathcal{M}\lvert_{\mathcal{S}}$	set of all structure matches in a pattern matching in ExpaRNA-P 40
$\mathcal{M} \setminus \mathcal{M}\lvert_{\mathcal{S}}$	set of all sequence matches in a pattern matching in ExpaRNA-P 40
$\text{score}((\mathcal{M}, \mathcal{S}))$	score of EPM $(\mathcal{M}, \mathcal{S})$ in ExpaRNA-P 40
$\tau(i,j,k,l)$	score contribution for base pair match $(ij \sim kl)$ of base pairs (i,j) and (k,l) in ExpaRNA-P 40
$c_1(i,k)$	score contribution for a match $(i \sim k)$ in a base pair match in ExpaRNA-P 40
$c_2(i,j,k,l)$	score contribution for structure match in a base pair match $(ij \sim kl)$ in ExpaRNA-P 40

$c_3(i,j,k,l)$	score contribution for stacking in a base pair match $(ij \sim kl)$ in ExpaRNA-P 40
α_1	weight factor for $c_1(i,k)$ in ExpaRNA-P 41
α_2	weight factor for $c_2(i,j,k,l)$ in ExpaRNA-P 41
α_3	weight factor for $c_3(i,j,k,l)$ in ExpaRNA-P 41
str-mm	structure mismatch score for relaxed EPMs in ExpaRNA-P 41
ψ	pseudo base pair match that matches the pseudo base pairs of the two sequences 41
$\text{parent}_{\mathcal{S}}(i \sim k)$	parent of a match in a pattern matching in ExpaRNA-P ... 41
$\text{parent}_{\mathcal{S}}(ij \sim kl)$	parent of a base pair match in a pattern matching in ExpaRNA-P 41
θ_2	threshold for unpaired in-loop probabilities 41
θ_3	threshold for base pair in-loop probabilities 41
$D(ij,kl)$	scores the best EPM enclosed by base pair match $(ij \sim kl)$ in ExpaRNA-P 44
$L^{ijkl}(j',l')$	scores the best EPM that is connected to the left ends of the base pair match in ExpaRNA-P 44
$G_{\text{A}}^{ijkl}(j',l')$	scores the best EPM where an arbitrary number of gaps was introduced in sequence A, in ExpaRNA-P 44
$G_{\text{AB}}^{ijkl}(j',l')$	scores the best EPM where an arbitrary number of gaps was introduced in sequence A and B in ExpaRNA-P 44
$H^{ijkl}(j',l')$	auxiliary matrix in ExpaRNA-P 44
$LR^{ijkl}(j',l')$	scores the best EPM that closes a gap left of the sequence or structure match in ExpaRNA-P 44
$F(j',l')$	scores the best EPM that is allowed to start at any point and ends at position (j',l') in ExpaRNA-P 44
$\text{wo-gap}^{ijkl}((\bar{x},\bar{y}),(i',k'))$	returns true if corresponding pair of sequence positions of matrix position $(\bar{x},\bar{y})$ and sequence positions i' and k' are directly adjacent; otherwise false 48
$\text{min-col}(j')$	minimal column for row j' when the `max-diff` constraint is used 53

max-col(j')	maximal column for row j' when the `max-diff` constraint is used 53
δ	maximal allowed deviation defined by `max-diff` 53
col-idx-begin$^{ijkl}(\bar{x})$	minimal column for row $\bar{x}$ in a sparsified matrix for base pair match $(ij \sim kl)$ when the `max-diff` constraint is used 53
col-idx-end$^{ijkl}(\bar{x})$	column after maximal column for row $\bar{x}$ in a sparsified matrix for base pair match $(ij \sim kl)$ when the `max-diff` constraint is used 53
Ψ_{ij}^{A}	weight of base pair (i, j) in sequence A 68
$\mathcal{A}_{RT}^{u}$	all parts of sequence-structure alignment $(\mathcal{A}, R, T)$ not covered by the structures, i.e. the unstructured part 68
score$((\mathcal{A}, R, T))$	score of $(\mathcal{A}, R, T)$ in PMcomp-derived approaches 68
$\bar{M}(i, j, k, l)$	scores the best alignment of subsequences $A_{i..j}$ and $B_{k..l}$ in original PMcomp recursion 68
$\bar{D}(ij, kl)$	scores the best subalignment enclosed by match of (i, j) and (k, l) including a score for matching the two base pairs in original PMcomp recursion 68
$\hat{M}^{ijkl}(j', l')$	scores the best alignment of subsequences $A_{i+1..j'}$ and $B_{k+1..l'}$ in PMcomp variant 68
$\hat{D}(ij, kl)$	analogous to $\bar{D}(ij, kl)$ for PMcomp variant 68
$\tilde{D}_S(ij, kl)$	analogous to $\bar{D}(ij, kl)$ for PARSE 69
$\tilde{M}^{ijkl}(j', l')$	analogous to $\hat{M}^{ijkl}(j', l')$ for PARSE 69
$\tilde{I}_{\mathrm{A}}^{ijkl}(j')$	scores a loop deletion enclosed by match of (i, j) and (k, l) in PARSE 69
$\tilde{I}_{\mathrm{B}}^{ijkl}(l')$	scores a loop insertion enclosed by match of (i, j) and (k, l) in PARSE 69
$\tilde{E}^{ijkl}(j', l')$	covers insertion case for affine gap cost (base $B_{l'}$ is inserted) in PARSE 70
$\tilde{F}^{ijkl}(j', l')$	covers deletion case for affine gap cost (base $A_{j'}$ is deleted) in PARSE 70
γ_{base}	gap cost for base insertion or deletion 70
γ_{loop}	gap cost for loop insertion or deletion 70

β_{base}	gap opening cost for base insertion or deletion 70
β_{loop}	gap opening cost for loop insertion or deletion 70
$D_S(ij, kl)$	scores the best subalignment enclosed by match of (i, j) and (k, l) *without* a score contribution for the match of the base pairs in SPARSE 75
$M^{ijkl}(\bar{x}, \bar{y})$	scores the best alignment of subsequences up to matrix position $(\bar{x}, \bar{y})$ enclosed by match of (i, j) and (k, l) in SPARSE 75
$I_{\text{A}}^{ijkl}(\bar{x})$	scores a loop deletion enclosed by match of (i, j) and (k, l) in SPARSE 75
$I_{\text{B}}^{ijkl}(\bar{y})$	scores a loop insertion enclosed by match of (i, j) and (k, l) in SPARSE 75
N'	represents an RNA structure (Nussinov variant), Sparsification in ADP 87
h	choice function, Sparsification in ADP 87
N	represents an RNA structure (Nussinov variant), Sparsification in ADP 89
S	represents an RNA structure that can be split into two parts, Sparsification in ADP 89
C	represents a closed RNA structure, Sparsification in ADP ... 89
$\backslash_h$	novel sparsification operator, Sparsification in ADP 92
C^c	contains the OCT fragments, Sparsification in ADP 92
$\bar{L}$	represents fragments that are unpaired on the left side, Sparsification in ADP 92
L^c	contains the STEP fragments, Sparsification in ADP 92