Proceedings of the ECCB’14 workshop on Computational Methods for Structural RNAs (CMSR’14) Strasbourg, France

Fabrice Jossinet, Université de Strasbourg
Yann Ponty, CNRS/Ecole Polytechnique
Jérôme Waldispühl, McGill University

September 7th, 2014
Preface

This volume contains the original papers and abstracts presented at CMSR’14: Computational Methods for Structural RNAs held on September 7, 2014 in Strasbourg. This event was hosted as an official workshop of ECCB’14: 13th European Conference on Computational Biology.

Ribonucleic acids (RNAs) play key roles in various aspects of the gene transcription and regulation processes, and are the focus of an ever-increasing interest in all areas of molecular biology. Deciphering the function of a non-protein coding RNA requires an intimate knowledge of its structure, motivating the development of structure-centric methods.

The relevant technology encompasses a wide variety of techniques ranging from biophysical simulations to exact calculation of mathematical models. Similarly, the definition of structure can be remarkably large, starting from classical secondary structures to multimeric systems created by inter-molecular interactions. Therefore, the concepts addressed in this workshop naturally spans a broad range of interests in biological research. The development of new sequencing technologies and high-throughput molecular probing experiments is also at the origin of a rapid increase in size and complexity of RNA data sets, opening new computational challenges to analyze them.

This workshop aims to gather researchers and students, developing or using computational methods for predicting RNA structures or analyzing their contribution to biological processes. It provides a forum for the dissemination of state-of-the-art methods and tools using discrete representations of folding landscapes in the context of big data. It contributes to establish best-practices towards a better support for large-scale data within structure-centric tools and algorithms.

The program consists of 4 selected papers, each submission being reviewed by 3 program committee members. The program also includes 3 highlight talks, and two keynote addresses, delivered by Mihaela Zavolan (Biozentrum, University of Basel, Switzerland) and Jan Gorodkin (Center for Non-Coding RNA in Technology and Health, University of Copenhagen, Denmark).

July 10, 2014
Paris, France

Fabrice Jossinet
Yann Ponty
Jérôme Waldispühl
Sponsors

Groupe De Recherche BioInformatique Moléculaire (GDR-BIM)
Molecular Bioinformatics CNRS research group

2014 European Conference on Computational Biology

Laboratoire d’Excellence NetRNA
NetRNA consortium

Institut National de Recherche en Informatique et Automatique

McGill School of Computer Science

Acknowledgments

The organizers wish to express their gratitude to:

- The ECCB’14 organizers, for hosting the event as an official workshop of the conference, and their continued help with all practical aspects of its organization;
- The CMSR steering committee, consisting of Peter Stadler (Univ. Leipzig, Germany), Eric Westhof (Univ. Strasbourg, France), and Ivo Hofacker (TBI Vienna, Austria) for their scientific guidance;
- The ‘EasyChair’ conference management system, for greatly alleviating the burden of the organizers throughout all stages of the selection process;
- The author (V. Verfaille) of the \LaTeX{}package ‘confproc’, which provided invaluable help during the typesetting of this document.
Steering Committee

Ivo Hofacker  Institute for Theoretical Chemistry, University of Vienna, Austria
Peter Stadler  Leipzig University, Germany
Éric Westhof  IBMC, Strasbourg, France

Program Committee

Rolf Backofen  Albert-Ludwigs-University Freiburg, Germany
Sarah Burge  Wellcome Trust Sanger Institute/EMBL-EBI, United Kingdom
Cedric Chauve  Simon Fraser University, Canada
Alain Denise  LRI - Univ. Paris-Sud/Inria Saclay, France
Christoph Flamm  University of Vienna, Austria
Christine Gaspin  INRA Toulouse, France
Christian Hoener Zu Siederdissen  Institute for Theoretical Chemistry, University of Vienna, Austria
Stefan Janssen  Universität Bielefeld, Germany
Fabrice Jossinet (co-chair)  Université de Strasbourg, France
Vincent Moulton  University of East Anglia, United Kingdom
Dmitri Pervouchine  Centre for Genomic Regulation, Spain
Yann Ponty (co-chair)  LIX Ecole Polytechnique/Inria Saclay, France
Jing Qin  University of Southern Denmark
Stefan Seemann  Center for non-coding RNA in Technology and Health, Denmark
Jerôme Waldispühl (co-chair)  University McGill, Canada
Sebastian Will  Leipzig University, Germany
Michal Ziv-Ukelson  Ben Gurion University of the Negev, Israel

Additional Review by Christina Otto (Albert-Ludwigs-University Freiburg, Germany).

Organizing Committee

Fabrice Jossinet  Université de Strasbourg, France
Yann Ponty  LIX Ecole Polytechnique/Inria Saclay, France
Jerôme Waldispühl  University McGill, Canada
Conference Program

Selected papers
1  Chaining Sequence/Structure Seeds for Computing RNA Similarity  
   Laetitia Bourgeade, Julien Allali and Cédric Chauve
13  Searching for alternate RNA structures in genomic sequences  
    Azadeh Saffarian, Mathieu Giraud and Hélène Touzet
25  Ab initio Prediction of RNA Nucleotide Interactions with Backbone $k$-Tree Model  
    Liang Ding, Xingran Xue, Sal Lamarca, Mohammad Mohebbi, Abdul Samad,  
    Russell L. Malmberg and Liming Cai
43  Observations on the Feasibility of Exact Pareto Optimization  
    Cédric Saule and Robert Giegerich

Research highlights
57  Recent Results on Three Problems in Comparative Structural RNAomics  
    Shay Zakov, Nimrod Milo, Tamar Pinhas, Sivan Yogev, Erez Katzenelson,  
    Eitan Bachmat, Yefim Dinitz, Dekel Tsur and Michal Ziv-Ukelson
59  Scalable structural clustering of local RNA secondary structures  
    Fabrizio Costa, Steffen Heyne, Dominic Rose and Rolf Backofen
62  The role of periodic mRNA secondary structure and RNA-RNA interactions  
    in biological regulation and complexity  
    Svetlana Shabalina, Aleksey Ogurtsov, Anna Kashina and Nikolay Spiridonov

Keynote addresses
65  Searching for SNPs disrupting RNA secondary structures  
    Jan Gorodkin
66  Deciphering the regulatory functions of miRNAs  
    Mihaela Zavolan

Index of Authors
67
Chaining Sequence/Structure Seeds for Computing RNA Similarity

Laetitia Bourgeade 1, Cédric Chauve 1,3, and Julien Allali 1,2

1 LaBRI, Université de Bordeaux, 341 cours de la Libération, Talence, France
2 Institut Polytechnique de Bordeaux, 146 Rue Léo Saignat, Talence, France
3 Department of Mathematics, Simon Fraser University, Burnaby (BC), Canada
{lætitia.bourgeade, julien.allali}@labri.fr{cedric.chauve}@sfu.ca

Abstract. We describe a new method to compare a query RNA with a static set of target RNAs. Our method is based on (i) a static indexing of the sequence/structure seeds of the target RNAs, (ii) searching the target RNAs by detecting seeds of the query present in the target, chaining these seeds in promising candidate homologs, then (iii) completing the alignment using an anchor-based exact alignment algorithm. We apply our method on the benchmark Bralibase2.1 and compare its accuracy and efficiency with the exact method LocaRNA and its recent seeds-based speed-up ExpLoc-P. Our pipeline RNA-unchained greatly improves computation time of LocaRNA and is comparable to the one of ExpLoc-P, while improving the overall accuracy of the final alignments.

1 Introduction

A major advance in molecular biology of the last decade has been the discovery that RNA molecules, especially non-coding RNAs (ncRNAs), are involved in many cellular processes such as the regulation of gene expression, splicing, signaling, . . . [16]. This is well illustrated by the growth of the Rfam database [7], whose content went from 15,255 RNAs, in 2002 (date of its creation) to 6,125,803 RNAs, in 2012 (last release). Moreover, several recent studies of the RNA structurome at the whole genome level have lead to the discovery of new families of ncRNAs and to a better understanding of the role of RNAs in the cell [20,11,18].

The general problem of annotating, classifying or clustering of RNA sequences is thus an important problem in computational biology, that relies on solving efficiently and accurately the following computational question: given an RNA query Q and a set of RNA sequence targets D, what are the members of D whose similarity with Q is large enough to indicate a potential relationship, either evolutionary and/or functional? For RNA genes, due to the importance of the structure in terms of biological function, it is natural to consider both the sequence and secondary structure when comparing genes. Most RNA comparison methods can be classified in two families: (i) tools requiring the knowledge of an RNA secondary structure, such as RNAforester [9] or Gardenia [4] to cite only two (see [3] for a thorough evaluation of such methods), and (ii) tools taking only RNA sequences as input and using covariance models or base pairing probabilities such as LocaRNA [20] or Infernal [13]. The first family of approaches relies on
Bourgeade, Chauve and Allali

the classical notions of edit distance and alignment. RNAs are modeled using tree-like structures and algorithms look either for a set of edit operations of optimal score that transforms the first RNA into the second one, or for an alignment maximizing the similarity. A cubic time complexity is the current reference for pairwise RNA structure comparison (see for example [21]), underlining the issue of using such approach directly when a large number of pairwise comparison is required. The second family of RNA pairwise comparison methods works directly on RNA sequences. The current reference method LocaRNA aligns RNA sequences based on the pairing probabilities for each sequence, computed from the partition function of the ensemble of all possible foldings into secondary structures, under the assumption of a free-energy based Boltzmann distribution on this ensemble and has a quartic time complexity. Approaches have been introduced to speed-up the alignment, at the expense of guaranteed optimality, such as Exp Loc-P/Exploc-P [14]; these methods rely on conserved sequence/structure motifs, called Exact Pattern Matches (EPM), that can be detected in quadratic time and are provided to LocaRNA as alignment constraints, thus breaking the alignment computation into smaller independent problems and reducing the overall computation time. Finally, a last set of tools aims at solving the classification problem, that asks to assign a given query RNA sequence to a set of predefined families, such as the Rfam. The Rfam classification engine Infernal starts by computing covariance models for families based on RNA sequences that are known to belong to them. Again, despite recent improvements this approach remains time consuming, which has motivated the development of filters such as RNA sifter [10], based on the abstract shape approach (see [15]).

In the present work, we address the general problem of the one-against-all RNA pairwise comparison, where a given query RNA \( Q \) is compared to unstructured set \( D \) of target RNAs. We introduce a new method, RNA-unchained, aimed at computing efficiently high quality alignments between \( Q \) and the members of \( D \). Our method is based on a classical principle in sequence comparison following four steps: seed indexing for the target set, seeds look-up in the index for the given query, seeds chaining between the query and the targets sharing common seeds, and finally exact anchor-based alignments. To evaluate RNA-unchained, we followed the approach of [14] and used the benchmark BRAliBase2.1, which is composed of set of reference pairwise alignments between ncRNA sequences. We measured the accuracy of the alignments obtained by RNA-unchained, using the Sum of Pair Scores statistics (SPS) [17]. We observe that we obtain alignments of quality comparable or better than LocaRNA, and consistently better than ExpLoc-P, in a time comparable to the time taken by ExpLoc-P.

2 Methods

The pipeline we describe takes as input a set \( D \) of RNA sequences and an RNA sequence query \( Q \), and aims at computing quickly candidate sequences of \( D \) that are similar to \( Q \), together with alignments between \( Q \) and these candidates. Our pipeline applies to the case where secondary structures are provided or not. It is composed of two elements: a static preprocessing stage for \( D \) and, for a given query \( Q \), a dynamic search in \( D \) for RNA similar to \( Q \).
Chaining Sequence/Structure Seeds for Computing RNA Similarity

**Preprocessing D.** This stage is static, i.e. is performed once for all. It consists in folding the sequences of D, each into one or several candidate RNA secondary structures, followed by extracting and indexing a set of seeds, defined as sequence/structure motifs of a given length.

**Querying D.** For a given query Q, its sequence is first folded into an RNA secondary structure and all the seeds it contains are generated. Then the index of seeds from D is searched to identify candidate sequences in D sharing motifs with Q. Next for each candidate from D, an optimal set of seeds that is compatible with the secondary structures of both Q and the candidate is extracted using a fast seeds chaining algorithm, and these seeds are used as anchors for a constrained alignment between the candidate and Q.

We now describe the details of our pipeline, starting with the modeling of RNA secondary structures and seeds.

### 2.1 Modeling RNA Secondary Structures and Seeds

**RNA sequence and secondary structure.** An RNA is a molecule composed of four nucleic acids usually symbolized by the alphabet \( \{A, C, G, U\} \). Pairs of bases in an RNA molecule can form hydrogen links, thus generating a spatial folding of the molecule forming its secondary structure. Here we consider pseudoknot-free RNA secondary structures, i.e. we assume that each base is involved in at most one base pair and that base pairs define a crossing-free planar structure. An RNA secondary structure can be encoded by an **arc-annotated sequence** (aa-sequence for short) \([\cdot, (\cdot)]\), and we rely on this modeling to describe our method. An aa-sequence \( A = (S, P) \) representing a pseudoknot-free RNA structure is composed of a sequence \( S \) of length \( |A| = n \) on the alphabet \( \{A, C, G, U\} \), representing the RNA primary structure (sequence) and of a well-parenthesized sequence \( P \) of length \( n \) on the alphabet \( \{., (, )\} \), representing the paired bases defining the secondary structure. For a sequence \( S \), we denote by \( S[i] \) the \( i \)\(^{th}\) symbol of \( S \) and by \( S[i, j] = S[i]S[i + 1] \ldots S[j] \) the factor of \( S \) of length \( j - i + 1 \) or starting at position \( i \), for any \( i \leq j \in \{0, \ldots, n - 1\} \). Similar definitions hold for \( P \).

**Seeds.** Seeds, defined as sequence/structure motifs, are used for two purposes in our pipeline. They are first aimed at detecting quickly candidates RNAs from D that share enough seeds with Q. In a second time, an optimal set of seeds that are compatible with the secondary structures of Q and the candidate is computed, and serves as anchors for the final alignment. So the definition of seeds should (1) allow a fast look-up in the indexing structure and (2) satisfy some compactness condition that makes them compatible with the chaining algorithm we use [2].

**Definition** \((l,d)\)-centered-seed (short name \((l,d)\)-cs) Let \( A = (S, P) \) be an aa-sequence of length \( n \). Let \( d \) and \( l \) be two integers such that \( 2d \leq l \). For a given \( i \in \{0, \ldots, n - 1\} \), the \((l,d)\)-cs of \( A \) in position \( i \), denoted by \( cs_i \), is the pair \((s, p)\) defined by \( p = P[i, i + l - 1] \) and \( s = S[i + d, i + l - d - 1] \).
Note that $s$ is a sequence of length $l - 2d$ and $p$ a sequence of length $l$, so a $(l,d)$-cs is not an aa-sequence as both sequences do not have the same length (see Fig. 1). It follows immediately that the maximal number of distinct $(l,d)$-cs is $3^d l^{l-2d}$. Furthermore, such seeds can be seen as spaced seeds [5] with no structural mismatch and possible nucleotide mismatches in a prefix and a suffix of length $d$ of the seed. Next, we define the notion of seed common to two aa-sequences, that we call a hit.

**Definition** (hit) Let $A_1 = (S_1, P_1)$ and $A_2 = (S_2, P_2)$ be two aa-sequences. Let $d$ and $l$ be two integers such that $2d \leq l$. A hit is an $(l,d)$-cs common to $A_1$ and $A_2$, i.e. a pair $(i,j)$ of integers such that:

- $0 \leq i \leq |A_1| - l - 1$ and $0 \leq j \leq |A_2| - l - 1$.
- $P_1[i,i+l-1] = P_2[j,j+l-1]$.
- $S_1[i+d,i+l-d-1] = S_2[j+d,j+l-d-1]$.

The score of a hit $S$ between two aa-sequences $S_1$ and $S_2$, composed of a conserved $(l,d)$-cs located at positions $i$ in $S_1$ and $j$ in $S_2$ is defined by:

$$
\text{score}(S) = \sum_{k=0}^{l-1} f(S_1[i+k], S_2[j+k])
$$

where $f(a,a) = 1$ and $f(a,b) = 0$ if $a \neq b$.

It follows that the score $s$ of $S$ satisfies $l - 2d \leq s \leq l$. For example on Fig. 2 the score of {"AA","((..))"} is 3.

![Fig. 1. Example of hits between two RNAs Q and T. We can notice that h_{q8} / h_{q9} and h_{q8} / h_{t9} are overlapping hits and h_{q8} / h_{q23} and h_{q8} / h_{t23} are crossing hits. So there is 6 hits \{h_{q6};h_{t6}),(h_{q7};h_{t7});(h_{q8};h_{t8});(h_{q8};h_{q9});(h_{q22};h_{t23});(h_{q22};h_{t23})\}.](image)

**2.2 Seed indexing and hits lookup**

The first key element in the method we present consists in indexing in a hash table all $(l,d)$-cs present in the RNA target set $D$ of interest, for given parameters $l$ and $d$. We denote by $\mathcal{F}_d^l$ this index. Comparing $Q$ with the RNAs from $D$ starts by searching in $\mathcal{F}_d^l$ the RNAs of $D$ having seeds present in $Q$. 
Chaining Sequence/Structure Seeds for Computing RNA Similarity

Indexing seeds. Given an aa-sequence \( A = (S, P) \) of size \( n \) and \( cs \) parameters \( l \) and \( d \), all \( k = n - l + 1 \) seeds of \( A \) are indexed in \( J^d_l \). To do this all computed \( (l, d)cs \) are converted in integers as follows: the \( cs \) encoding for a \( (l, d)cs \) on \( A \) at \( d^{th} \) position is defined by

\[
S_{Value}(A, i, l, d) = 4^{l-2d} + \sum_{j=1}^{i+l-1}(encode(P_j) \times 3^{i+l-1-j}) + \sum_{j=i+l}^{i+l+d-1}(encode(S_j) \times 4^{i+l-d-1-j})
\]

with \( encode : A = 0 ; C = 1 ; G = 2 ; U = 3 ; . . . = 0 ; (= 1 ; ) = 2 \)

Given an integer \( x \), \( J^d_l [x] \) will contain all occurrences of the \( cs \) which \( S_{Value} \) is \( x \), that is

\[
J^d_l [x] = \{ (A, i) | S_{Value}(A, i, l, d) = x \}
\]

For example, with the aa-sequences of Fig. 1, the integer associated to the \( (l, d)cs \) \(("AA", "(\ldots)"\) is 5312, and \( J^d_8 [5312] = \{(Q, 8), (Q, 22), (T, 8), (T, 23)\}\).

Inserting all \( cs \) of an RNA is done in linear time using a sliding window of length \( l \).

Index look-up. Given a query \( Q \), the search for aligning it with the target RNAs from \( D \) starts by computing all the seeds of \( Q \), for a given pair \((l, d)\) of parameters, then searching \( J^d_l \) for all RNAs from \( D \) that have seeds present in \( Q \). For a given RNA \( T \) from \( D \), let \( L^d_j [Q, T] \) be the set of all \( (l, d) \) \(-cs\) common to both \( Q \) and \( T \), i.e. hits:

\[
L^d_j (Q, T) = \{ (i, j) | S_{Value}(Q, i, l, d) = S_{Value}(T, j, l, d) \}
\]

For example, using the same aa-sequences as in Fig. 1, \( L^d_j (Q, T) = \{ (0, 0), (8, 8), (8, 23), (22, 8), (9, 9), (22, 23)\}\). This phase is done using a standard hash-table look-up using the integer associated to each seed as key. The time required to compute the \( L^d_j [Q, T] \) is linear in the size of the query and the overall number of hits. RNA-unchained offers the option to reduce the set of candidates based on the number of hits.

Hits/seeds optimization. Preliminary experiments showed that hits that do not contain both structural signal were more likely to be false positive. So in order to obtain more stringent seeds, the hits in \( L^d_j (Q, T) \) can be filtered to keep only the ones with two types of RNA structural symbols structure (right base of base pair and left base of base pair) which correspond to seeds that comprise the hairpin-loops: ( ) and stems junction: ( ) motifs, that are known to be well preserved and important structural patterns to detect secondary structure similarity [1].
Fig. 2. Compared to Fig. 1 3 hits are lost because of structural composition, crossing and overlapping. So this example shows the final hits between $Q$ and $T$.

2.3 Chaining Algorithm and Anchors

The core of our approach to align $Q$ with the target set $D$ is to first compare $Q$ and a member $T$ of $D$ using solely their hits. To do so, we use a recent efficient algorithm for chaining seeds developed in [2], followed by a stage where gaps between seeds are completed using the LocaRNA algorithm. The first steps consists in extending seeds defining hits to account for base-pairing given by the considered RNA secondary structures.

For example on Fig. 3, the last parenthesis of $h_{t9}$ leads to the extension with the opening parenthesis of $Q$ at position 6.

Note that the corresponding seeds might not be contiguous along the sequences (as illustrated in Fig. 3). It follows that they do not satisfy the definition of EPM; however they satisfy the definition of seeds introduced in [2]. Given a set of $k$ extended hits for two RNAs $Q$ and $T$, an anchor is a subset of hits such that, first the corresponding seeds are non-overlapping in both $Q$ and $T$, second the seeds in both RNAs are compatible in terms of secondary structure (see chain definition in [2]). The score of an anchor $C$ is the sum of the scores of the seeds defining the hits it contains, and the chaining score between $Q$ and $T$ is the maximum score of an anchor, taken among all anchors between $Q$ and $T$ (called an optimal anchor). The algorithm we use computes an optimal anchor in time $O(k^2 \log k)$, where $k$ is the number of hits. In our running example, an anchor is composed of the hits $\{(Q, 9), (T, 9)\}, \{(Q, 22), (T, 23)\}$ (see Fig. 2). At the end of the chaining stage, we thus have, for each RNA $T$ from $D$ a set of hits between $Q$ and $T$ that forms an optimal anchor $\mathcal{A}(Q, T)$. We call gaps the segments of the RNAs $Q$ and $T$ that are not involved in the anchor.

2.4 Anchor extension

Prior to aligning the gaps of $Q$ and $T$ with an exact but more costly algorithm, we perform a phase of seeds extension aimed at reducing the gap size and compensating the fact that initial seeds are of bounded length. First, each hit of the anchor between the two RNAs $Q$ and $T$ is extended on both sides based on exact sequence similarity. As an example, on Fig. 3 the hit $(h_{t23}, h_{q22})$ is extended to the left by two nucleotides.

Next, still prior to the gaps alignment, we fill the remaining gaps using an adapted Longest Common Subsequence (LCS) considering triplet of letters (see Fig. 3 for an...
Chaining Sequence/Structure Seeds for Computing RNA Similarity

example) to avoid irrelevant constraints. This improvement has meaning only if the cover of the anchor is large enough. So we fix a threshold of $4l$ bases matches.

$Q \begin{array}{cccccccccccccccccc}
\text{G} & \text{G} & \text{C} & \text{U} & \text{A} & \text{A} & \text{U} & \text{C} & \text{G} & \text{A} & \text{A} & \text{C} & \text{G} & G & A & G & C & U & A & U & C & C
\end{array}$

$T \begin{array}{cccccccccccccccccc}
\text{C} & \text{G} & \text{C} & \text{U} & \text{U} & \text{A} & \text{C} & \text{G} & \text{A} & \text{A} & \text{C} & \text{G} & G & A & C & G & A & C & G & A & C & G
\end{array}$

Fig. 3. The matches between $Q$ and $T$ are left extended only for $h_{22}$ and $h_{23}$ and gaps are filled thanks to LCS computation.

2.5 Anchor-constrained alignment

Finally, for each candidate homolog $T$, the gaps defined by the anchor between $Q$ and $T$ are aligned using the exact algorithm LocaRNA where the anchor is provided as a set of constraints (see Fig. 4).

$Q \begin{array}{ccccccccccccccccccccccc}
\text{G} & \text{G} & \text{C} & \text{U} & \text{A} & \text{A} & \text{U} & \text{C} & \text{C} & \text{G} & \text{A} & \text{A} & \text{C} & \text{G} & A & G & C & U & A & U & A & G & A & G & A & A & U & C & C
\end{array}$

$T \begin{array}{ccccccccccccccccccccccc}
\text{C} & \text{G} & \text{C} & \text{U} & \text{U} & \text{A} & \text{C} & \text{G} & \text{A} & \text{A} & \text{C} & \text{G} & A & G & C & A & C & G & A & C & G & A & A & C & G & A & C & G
\end{array}$

Fig. 4. The anchor of our example, seen as sequence constraints.

3 Results

In order to assess the ability of RNA-unchained to provide accurate alignments, we applied on the set of reference RNA alignment provided by the benchmark Bralibase2.1 [19], composed of 8,976 pairwise alignments, classified into 36 families.

We analyzed this benchmark with RNA-unchained using several sets of options, LocaRNA used as a reference exact alignment tool and ExpLoc-P, a seed-based speed-up of LocaRNA. In order to reproduce the results on Bralibase2.1 shown in [14], we obtained from the authors of ExpLoc-P the corresponding code and parameters, and we ran ExpLoc-P with these optimized parameters.
In addition to these existing methods, we ran RNA-unchained with the following default parameters: (1) for each RNA sequence, its MFE secondary structure was obtained using RNafold [12,22], (2) the parameters \( l \) and \( d \) for seeds were chosen to be \( l = 9 \) and \( d = 1 \), after exploring a wide range of possible values for these parameters (see discussion below). We denote this default RNA-unchained version 91MFE. In order to assess the impact of adding stringency criterion to the seed selection process, as well as the impact of the anchor extensions methods described in the 2 section, we ran RNA-unchained with additional options. (1) 91r2: only (9,1)cs containing two-types parenthesis are conserved. (2) 91r2fb91epcLCS36 : 91r2 with seeds optimization, i.e. if there is no 91r2 seed consider 91MFE seeds, and anchor optimization. So altogether, we show the results of two reference programs (LocaRNA, ExpLocPOpt) and three versions of RNA-unchained (91MFE, 91r2 and 91r2fb91epcLCS36).

To compare the obtained alignments with the reference alignments from Bralibase2.1, we use the SPS statistics (Fig. 5). Given a reference alignment \( r \) of length \( l_r \) and a computed alignment \( e \) of length \( l_e \), the SPS is defined by the ratio \( \frac{SP^r}{l_r} \) where \( SP^r \) is the number of pairs \((i,j)\) where position \( i \) and \( j \) of the aligned sequences form a match in both \( r \) and \( e \). In addition, we also consider the coverage in percent of the input RNA by the anchors, defined as the ratio between the number of bases belonging to anchors by the length of the RNAs (Fig. 6). This statistic is important to evaluate the impact of anchors, both in terms of computation time and of accuracy, as a high coverage by wrong hits will mechanically result in a low SPS, while a very low coverage by high confidence hits might not result in a significant gain of computation time. Finally, we display our result according to the similarity between the pairs of compared RNAs, where the similarity value \( \text{Sim}(Q, T) \) between a query \( Q \) and a target \( T \) is defined, from the reference alignments, as follows:

\[
\text{Sim}(Q, T) = \frac{\sum_{i=0}^{l_r-1} f(Q'[i], T'[i])}{|Q'|} 
\]

where \( Q', T' \) are the aligned sequence derived from \( Q, T \) and \( f(a,a) = 1 \) and \( f(a,b) = 0 \) if \( a \neq b \). Both Fig. 5 and 6 present the number of alignments per similarity level (right scale).

We also show the computation time of the different methods that we considered. Note that the difference between the running time of LocaRNA and ExpLocPOpt is not as important as shown in [14], but the computation time of ExpLocPOpt is comparable between our experiments and [14]. A first point we can notice is that all methods perform well and with relatively similar behaviour, but for reference alignment between pairs of RNAs that exhibit a similarity in the range [0.6,0.8] (i.e. 60%-80%), where RNA-unchained used with stringent grains composed of at least two kinds of structural elements, performs better, and even obtains better results than LocaRNA, although LocaRNA is guaranteed to obtain alignment with better alignment scores. This shows that adding high quality anchors, even if they cover a smaller part of the considered RNA sequences (see Fig. 6, 91r2), can improve significantly the alignment quality. However, being too stringent in defining hits results in a coverage that is relatively low, which has for consequence that many alignments are constraint-free and rely purely on LocaRNA. Note however, that the low covering hits selected with this method still results
Chaining Sequence/Structure Seeds for Computing RNA Similarity

Fig. 5. SPS value for all 5 methods applied to the Bralibase2.1 benchmark.

in a significant accuracy improvement over LocaRNA in the range [0.6,0.8]. On the opposite, 91MFE has a good coverage meaning that LocaRNA takes great advantage on alignment computation as its computation time is divided by two. However, the accuracy of the alignments can be significantly lower than with the other methods, as some hits with low structural information are false positive that can be selected by the chaining, thus misleading LocaRNA in the final anchor-based exact alignment phase. Finally, the two optimizations we propose, the seed optimization (see paragraph 2.2) and the LCS-based anchor extension (see section 2.4) improves significantly the coverage by the anchor (see method 91r2/fb91LCS36) without impacting too much the alignment accuracy. This is reflected by the SPS values for 91r2/fb91LCS36 that are accuracy results while the coverage is the best one achieved by our versions. As a result, the gain in terms of alignment time compared to LocaRNA is maximal (divided by two).

Regarding seeds parameters, various values of \(l\) and \(d\) from \((5,1)\) to \((10,2)\) were tested before settling for the combination \(l = 9\) and \(d = 1\). The experiments we carried to explore these parameters (results not shown) indicated clearly that seeds with a small conserved sequence lead to many false positive seeds, while large conserved structures, especially with the additional requirement of two types of structural elements, lead to a very low coverage, and so a higher computational time in the gap alignment phase. In
Fig. 6. Coverage of RNA sequences by anchors in percent for all 4 seeds-based methods (LocaRNA alone does not use seeds) applied to the Bralibase2.1 benchmark.

general, these experiments show a relatively consistent pattern of correlated increasing coverage / decreased SPS.

4 Discussion

Summary. The main contribution we presented in this paper is a complete pipeline for the one-against-all RNA pairwise comparison, based on the notions of seeds, seeds index and seeds chaining. The key points are: a seed model describing both primary and secondary structure elements and a fast (sub-cubic) seeds chaining algorithm. The ability to index quickly and retrieve efficiently common seeds between a query and a set of targets is an important point of our method, that scales well as the main memory consumption of the index is determined by the number of keys in the hash table, which depends only on the seeds parameters \((l, d)\), as the data itself grows linearly with the cumulated size of the target RNAs. Our experiments using the benchmark Bralibase2.1 show clearly that RNA-unchained obtains results that are more accurate than current state-of-the art methods, with comparable computation times (and probably better computation times in the C++ next release).
Chaining Sequence/Structure Seeds for Computing RNA Similarity

<table>
<thead>
<tr>
<th>Computation time</th>
<th>Hits/chaining</th>
<th>Gaps alignment</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>LocaRNA</td>
<td>0</td>
<td>9,022</td>
<td>9,022</td>
</tr>
<tr>
<td>ExpLocPOpt</td>
<td>1,492</td>
<td>6,070</td>
<td>7,562</td>
</tr>
<tr>
<td>91MFE</td>
<td>3,386</td>
<td>4,563</td>
<td>7,949</td>
</tr>
<tr>
<td>91r2</td>
<td>3,157</td>
<td>6,242</td>
<td>9,399</td>
</tr>
<tr>
<td>91r2/191epcLCS36</td>
<td>3,283</td>
<td>4,510</td>
<td>7,793</td>
</tr>
</tbody>
</table>

Table 1. Computation times (in seconds). The time required to build the index is not included but takes less than 1 minute. Experiments were performed on a server with double Intel Xeon 3.3GHz processor. The seeds indexing, hits look-up and chaining are implemented in Java.

Seeds model and parameters. The seed model we introduce differ significantly from the ExpaRNA model while EPMs are designed as connected subgraphs of the RNA secondary structure. It is interesting to notice that our seeds model (with the seeds and anchors extensions) provide a coverage of the RNA sequences that is comparable to the one obtained with EPMs (results not shown). This shows that both models probably are able to capture important conserved structural features. However, we can notice that in the similarity range 60%-80%, where RNA-unchained outperforms ExpLoc-P and LocaRNA in terms of SPS, we observe a significant difference. More generally, our work and the line of work centered on EPM suggest that the general seeds chaining approach deserves to be explored, both in terms of seed models and chaining algorithms. In particular, unlike sequence seeds, that have been deeply studied, formal studies of RNA seeds, including statistical aspects, are lacking.

Secondary structure. A major difference between our approach and the LocaRNA/ExpLoc-P lies in the way the secondary structure of RNA is accounted for. We explored several intermediate approaches, based on sampling RNA secondary structures using RNAsubopt and RNAshapes or based on keeping alignment with best score when using several suboptimal structures, but found that the MFE still provided the best accuracy results while minimizing the computation time (results not shown). This apparent concordance between two very different approaches suggests again that the notion of RNA structural seeds still deserves to be further studied.

Gap filling and chaining. An important aspect of RNA-unchained concerns the use of LocaRNA for the anchor-based gaps alignment. As gaps are segments where conserved structural motifs are absent, this allows to reduce the impact of the choice of the MFE, that is used only to detect seeds and compute the anchor, and is likely to be one of the reasons that explains the concordance between both approaches. However, this part of our pipeline is still the most costly in terms of computation time. As suggested by the results obtained with our LCS anchor-extension, a hierarchical/iterative approach, that would consider less conserved sequence and/or structure motifs detected within a given gap, and thus help again to reduce the segments on which an exact alignment algorithm is used, could be an efficient approach. Taking a somewhat extreme point of view, one could even ask if, in applications where exact alignments are not needed, the approach...
Bourgeade, Chauve and Allali described above, limited to the computation of extended anchors, possibly completed by a quick way to evaluate the similarity between short gaps in RNA sequences, would not be sufficient.

References

Searching for alternate RNA structures in genomic sequences

Azadeh Saffarian, Mathieu Giraud, and Hélène Touzet
LIFL (UMR 8022 CNRS, University of Lille) and Inria Lille, France

Abstract. We introduce the concept of RNA multi-structures, that is a formal grammar based framework specifically designed to model a set of alternate RNA secondary structures. Such alternate structures can either be a set of suboptimal foldings, or distinct stable folding states, or variants within an RNA family. We provide several such examples and propose an efficient algorithm to search for RNA multi-structures within a genomic sequence.

1 Introduction

Structural RNAs play a wide range of roles in the cell of all organisms. In many RNA families the spatial architecture of the molecule is an important component of its function [1, 9]. This spatial architecture is mainly built upon the set of base pairings, that is captured in the secondary structure. Over the years, a great number of computational methods have been proposed to model consensus secondary structures. In many cases however, the signature for an RNA family cannot be compiled into a single consensus structure, and is mainly given by a set of alternate secondary structures. For example, certain classes of RNAs adopt at least two distinct stable folding states to carry out their function. This is the case of riboswitches, that undergo structural changes upon binding with other molecules [5], and recently some other RNA regulators were proven to show evolutionary evidence for alternative structure [11]. The necessity to take into account multiple structures also arises when modeling an RNA family with some structural variation across species, or when it comes to work with a set of predicted suboptimal foldings. The objective of this paper is to propose a model to deal with such set of alternate secondary structures. We introduce the formal concept of RNA multi-structures, that represent a set of alternate RNA secondary structures in a compact and non redundant way. The definition uses formal grammars. Formal grammars have been applied extensively to the problem of RNA folding, aligning and homology searching [6, 8, 10]. Compared to other tools from language theory, such as deterministic finite automata and HMMs, they are the method of choice because they are able to model long-range interactions. Here we exploit the power of this formalism to encode alternative foldings. The paper is organized as follows. In Section 2, we briefly recall some basic definitions. Section 3 introduces RNA multi-structures. Section 4 presents some examples to illustrate the utility of the concept. Lastly, in Section 5, we describe an algorithm to search for a RNA multi-structure in a genomic sequence.
Throughout this paper, we see RNA secondary structures as pure combinatorial objects, composed of helices. Assume we have a linear sequence $S$ provided with totally ordered positions. An interval is a pair of positions $\langle x, y \rangle$ of $S$ such that $x \leq y$. The interval $\langle x, y \rangle$ precedes the interval $\langle x', y' \rangle$ if $y < x'$. A helix $h$ is a pair of intervals $(h_1, h_2)$ such that $h_1 = \langle x, y \rangle$ precedes $h_2 = \langle x', y' \rangle$. So a helix is characterized by four positions on $S$, denoted $h.5\text{start}$, $h.5\text{end}$, $h.3\text{start}$, $h.3\text{end}$ for $x$, $y$, $x'$ and $y'$ respectively. We say that the helix $g$ is nested in the helix $h$ if $h.5\text{end} < g.5\text{start}$ and $g.3\text{end} < h.3\text{start}$. We say $g$ is juxtaposed with $h$ if $g.3\text{end} < h.5\text{start}$. A secondary structure is a set of helices that are all pairwise nested or juxtaposed. Note that in general any relation between any two helices is allowed: They can overlap, form a pseudoknot, or be included in each other.

![Fig. 1. Human mitochondrial tRNA sequence (source FJ004809.1/12137-12205 – RFAM, RF0005 [3]). (a) Helices are numbered from 1 to 8 from 5' to 3', and are written in Vienna bracket-dot format. (b) Helix graph. (c) Directed helix graph: $h \rightarrow g$ means that $g$ is juxtaposed with $h$, and $h \leftarrow g$ means that $g$ is nested in $h$.](image)
Searching for alternate RNA structures in genomic sequences

3 RNA multi-structures

Before proceeding to the formal definition, we illustrate our motivation with a simple example taken from the Human mitochondrial tRNA sequence. This sequence is 69nt long and has eight thermodynamically stable helices, which are represented in Figure 1-(a). We denote this set of helices $\mathcal{H}_0$. The elements of $\mathcal{H}_0$ can combine to form a variety of secondary structures. The question that we want to address is: How can we encode this set of all secondary structures, or a given subset of these secondary structures, into a single compact data structure? Furthermore we want to take advantage of the redundancy between the structures, since the structures may share some structural modules, and we want that the data structure can be efficiently used for further queries.

In first approach, the set of helices can be represented by a graph, called the helix graph, whose vertices are helices, and there is an edge between every two vertices if helices are either juxtaposed, or nested. This graph is represented in Figure 1-(b). In this graph, the set of all secondary structures is exactly the set of cliques. However, this representation does not convey the necessary information to understand the topology of the set of helices. A better approach is to classify edges in two categories. This is what is done in the directed graph of Figure 1-(c). Plain arcs are for juxtaposed helices, and dotted arcs are for nested helices. Arcs are oriented in the same direction as the $3\text{start}$ positions on the underlying sequence $S$. In this graph, any secondary structure can be seen as a selection of plain paths, such that any two plain paths are linked by dotted paths. Each plain path characterizes a flat structure.

Definition 1. A set of helices $w$ is a flat structure if, for any two distinct helices $h$ and $g$ in $w$, $h$ and $g$ are juxtaposed.

It is clear that any secondary structure can be partitioned into flat structures. We say that a flat structure $w$ is nested into a helix $h$ if all helices of $w$ are nested into $h$. We define a multi-structure as a combination of flat structures, generated by a tree grammar.

Definition 2. A multi-structure is a pair $\mathcal{M} = (\mathcal{H}, \mathcal{G})$, where $\mathcal{H}$ is a set of helices and $\mathcal{G}$ is a tree grammar. The grammar alphabet contains a start symbol $S$, a binary terminal symbol $\circ$, and for each helix $h$ of $\mathcal{H}$, a non-terminal symbol $H$ and a terminal symbol $h$. All its productions are of the form

\begin{align*}
(1) & \quad S \to H_1 \circ \ldots \circ H_q \\
(2) & \quad H \to h(H_1 \circ \ldots \circ H_q) \\
(3) & \quad H \to h
\end{align*}

where $h_1 \circ \ldots \circ h_q$ is any flat structure on $\mathcal{H}$ in (1), and a flat structure that is nested in $h$ in (2).

In the definition of $\mathcal{G}$, $H_1 \circ \ldots \circ H_q$ indicates which plain paths from the helix graph are authorized in the multi-structure, and $h(\ldots)$ which dotted paths are authorized in the multi-structure. The $\circ$ symbol allows to generate trees of arbitrary arity. An instance of the multi-structure $\mathcal{M}$ is any structure recognized by its grammar $\mathcal{G}$. It is straightforward to verify that each instance of a multi-structure is a secondary structure.
4 Examples of RNA multi-structures

4.1 Example on mitochondrial tRNA (cont’d)

We continue with our motivating example with Human mitochondrial tRNA sequence introduced in Figure 1. If we want to have all possible secondary structures for the helix set \( \mathcal{H}_0 \), productions for \( \mathcal{H}_0 \) are all possible rules that respect conditions (1), (2) or (3) in Definition 2.

\[
S \rightarrow H_1 \mid H_2 \mid H_2 \circ H_5 \mid H_2 \circ H_5 \circ H_8 \mid H_2 \circ H_6 \mid H_2 \circ H_7 \mid H_2 \circ H_8 \mid H_3 \mid H_4
\]
\[
H_1 \rightarrow h_1(H_2) \mid h_1(H_2 \circ H_5) \mid h_1(H_2 \circ H_5 \circ H_8) \mid h_1(H_2 \circ H_6) \mid h_1(h_1(H_2 \circ H_7)) \mid h_1(H_4 \circ H_7) \mid h_1(H_4 \circ H_8)
\]
\[
H_3 \rightarrow h_3(H_4) \mid h_3(H_4 \circ H_7) \mid h_3(H_5) \mid h_3(H_7) \mid h_3
\]
\[
H_4 \rightarrow h_4
\]
\[
H_5 \rightarrow h_5
\]
\[
H_6 \rightarrow h_6(H_7) \mid h_6
\]
\[
H_7 \rightarrow h_7
\]
\[
H_8 \rightarrow h_8
\]

By construction, the language of this grammar is exactly the set of all secondary structures on \( \mathcal{H}_0 \). The reader is invited to check that it contains 43 structures.

We take this example one step further and address the problem of building a multi-structure for all locally optimal secondary structures on \( \mathcal{H}_0 \). A locally optimal secondary structures is a secondary structure to which it is not possible to add a supplementary helix \([4, 12]\). In other words, it is a maximal clique in our helix graph of Figure 1. In \([12]\), we have proved that locally optimal secondary structures are exactly the secondary structures that are partitioned into some maximal flat structures (called maximal for juxtaposition structures), corresponding to some maximal plain paths in the helix graph. Here this result allows to define \( \mathcal{H}_1 \).

\[
S \rightarrow H_1
\]
\[
H_1 \rightarrow h_1(H_2 \circ H_5 \circ H_8) \mid h_1(H_2 \circ H_6) \mid h_1(H_3) \mid h_1(H_4 \circ H_8)
\]
\[
H_2 \rightarrow h_2
\]
\[
H_3 \rightarrow h_3(H_4 \circ H_7) \mid h_3(H_5)
\]
\[
H_4 \rightarrow h_4
\]
\[
H_5 \rightarrow h_5
\]
\[
H_6 \rightarrow h_6(H_7)
\]
\[
H_7 \rightarrow h_7
\]
\[
H_8 \rightarrow h_8
\]

The number of instances is reduced to 5: \( h_1(h_2 \circ h_5 \circ h_8) \), \( h_1(h_2 \circ h_6(h_7)) \), \( h_1(h_3(h_4 \circ h_7)) \), \( h_1(h_3(h_5)) \), \( h_1(h_4 \circ h_8) \). These instances correspond to all maximal cliques in the helix graph: \{1, 2, 5, 8\}, \{1, 2, 6, 7\}, \{1, 3, 4, 7\}, \{1, 3, 5\}, \{1, 4, 8\}. As expected, none are included in each other.

Lastly, multi-structures can be used to encode a pre-defined set of secondary structures. This time, we start from a set of candidate secondary structures with low free energy level. We ran the mfold software with 20% suboptimality \([13]\) on our tRNA sequence, and get four suboptimal structures, shown in Figure 2. From this set of structures, we extracted the set of helices, that appears to be the same helix set as \( \mathcal{H}_0 \). To
Searching for alternate RNA structures in genomic sequences

Fig. 2. Mfold output for the Human mitochondrial tRNA sequence

build the productions of the grammar $\mathcal{G}_2$, we consider for each helix $h$ the flat structure composed of all helices connected to the multibranch loop closed by $h$.

\[
\begin{align*}
S & \rightarrow H_1 \\
H_1 & \rightarrow h_1(H_2 \circ H_3 \circ H_6) | h_1(H_5) \\
H_2 & \rightarrow h_2 \\
H_3 & \rightarrow h_3(H_4 \circ H_7) | h_3(H_5) \\
H_4 & \rightarrow h_4 \\
H_5 & \rightarrow h_5 \\
H_6 & \rightarrow h_6(h_7) \\
H_7 & \rightarrow h_7 \\
H_8 & \rightarrow h_8
\end{align*}
\]

Compared to $\mathcal{G}_1$, the production $H_1 \rightarrow h_1(H_4 \circ H_6)$ is missing since none of the four suboptimal structures contains this combination of helices. This new multi-structure has four distinct instances, that correspond exactly to the four suboptimal secondary structures of Figure 2: \{1, 2, 6, 7\}, \{1, 3, 5\}, \{1, 2, 5, 8\} and \{1, 3, 4, 7\}. This construction is guaranteed to give all input structures. We will see in the following example that it can happen that some new instances are created by the grammar.

4.2 Multi-structures for bacterial RNAse P RNAs

In this second example, we explain how to model an RNA family with a multi-structure. Bacterial RNAse P RNAs fall into two major classes that share a common catalytic core, but also show some distinct structural modules: type A (represented by *Escherichia coli* in Figure 3) is the common and ancestral form found in most bacteria, and type B (represented by *Bacillus subtilis* in Figure 3) is found in the low-GC content gram-positive bacteria [7]. We explain how to gather the two structures in a same multi-structure. Following the nomenclature of the RNase P database, helices are labeled $P_5 - P_{18}$ from 5' to 3' in the *E. coli* structure. Helices $P_5$ and $P_6$ form pseudoknots and do not belong to the secondary structure. *B. subtilis* features some additional helices, denoted $P_{8.1}$, $P_{10.1}$, $P_{15.1}$ and $P_{19}$, and lacks helices $P_{13}$, $P_{14}$, $P_{16}$ and $P_{17}$. The corresponding grammar is obtained by taking all flat structures of each of the two consensus secondary structures for type A and type B.
Saffarian, Giraud and Touzet

\[ S \rightarrow P_1 \\
P_1 \rightarrow p_1(P_2) \mid p_1(P_3) \mid p_1(P_2 \circ P_19) \\
P_2 \rightarrow p_2(P_1) \mid p_2(P_3 \circ P_15 \circ P_15 \circ P_15 \circ P_18) \\
P_3 \rightarrow p_3 \\
P_5 \rightarrow p_5(P_3 \circ P_7) \mid p_5(P_7) \\
P_{5.1} \rightarrow p_{5.1} \\
P_7 \rightarrow p_{7}(P_8 \circ P_{10}) \\
P_8 \rightarrow p_8 \\
P_9 \rightarrow p_9 \\
P_{10} \rightarrow p_{10}(P_{10.1} \circ P_{11}) \mid p_{11} \\
P_{10.1} \rightarrow p_{10.1} \\
P_{11} \rightarrow p_{11}(P_{12}) \mid p_{11}(P_{12} \circ P_{13} \circ P_{14}) \\
P_{12} \rightarrow p_{12} \\
P_{13} \rightarrow p_{13} \\
P_{14} \rightarrow p_{14} \\
P_{15} \rightarrow p_{15}(P_{16}) \mid p_{15} \\
P_{15.1} \rightarrow p_{15.1} \\
P_{16} \rightarrow p_{16}(P_{17}) \\
P_{17} \rightarrow p_{17} \\
P_{18} \rightarrow p_{18} \\
P_{19} \rightarrow p_{19} \]

Green non-sulfur Bacteria RNase P RNAs are known to show some notable variation against the forms A and B. The majority of them (represented here by \textit{H. auriantacu} in Figure 3) are of the type A class, except for the structural module \textit{P}_{18}/\textit{P}_{15.1} that is instead quite similar to that of the type B and for the helix \textit{P}_{19}. One exception is represented by the sequence of \textit{T. roseum}, which has independently converged with the class B RNAs (presence of helices \textit{P}_{10.1} and \textit{P}_{15.1}, and absence of helices \textit{P}_{13}, \textit{P}_{14} and \textit{P}_{19}). However, \textit{T. roseum} RNA retains \textit{P}_{18}/\textit{P}_{17} and does not contain \textit{P}_{1.1} [7] (see Figure 3). Interestingly, these two variants appear as instances of the multi-structure designed to encode solely types A and B. It means that in this example, these two new structures created by the multi-structure are functional and have been selected by the evolution.

5 Searching multi-structures in genomic sequences

We now turn to the following problem: Given a multi-structure \( M = (H', G) \) and a text such as a genomic sequence, how to identify occurrences of \( M \) in a text. For that, we assume that the text is given by a list of putative helices \( K \), that have been obtained by preprocessing the genomic sequence. We define formally what is an occurrence of a multi-structure in such a text.

\textbf{Definition 3.} Let \( H \) and \( K \) be two sets of helices. A helix mapping from \( H \) to \( K \) is an injective function \( \phi : H \rightarrow K \) such that, for any two helices \( h \) and \( h' \) in \( H \), if \( h \) is nested in \( h' \), then \( \phi(h) \) is nested in \( \phi(h') \), and if \( h \) is juxtaposed with \( h' \), then \( \phi(h) \) is juxtaposed with \( \phi(h') \).

\textbf{Definition 4.} Let \( M = (H', G) \) be a multi-structure and \( K \) be a text defined by a set of helices. An occurrence of \( M \) in \( K \) is a pair \((H', \phi)\), where \( H' \) is a subset of \( H \) and \( \phi \) is a helix mapping from \( H' \) to \( K \).
Searching for alternate RNA structures in genomic sequences

Fig. 3. Secondary structures for bacterial RNase P RNA (source [2]).
This definition allows for approximate search with errors, since \( \mathcal{H}' \) is not required to contain all helices of the instances of the multi-structure. Note also that in this general definition, we do not specify whether one instance or all instances of the multi-structure should appear in the text. This gives rise to two distinct searching problems, that are formalized as follows.

**Simple occurrence problem:** Let \( e \) be a natural number. Find in the text \( K \) all occurrences of \( \mathcal{M} \) with at most \( e \) errors, where the number of errors of an occurrence \( (\mathcal{H}', \phi) \) is defined as the minimal number of helices appearing in some instance of \( \mathcal{M} \) and that are not in \( \mathcal{H}' \).

**Universal occurrence problem:** Let \( e \) be a natural number. Find in the text \( K \) all occurrences of \( \mathcal{M} \) with at most \( e \) errors, where the number of errors of an occurrence \( (\mathcal{H}', \phi) \) is defined as the maximal number of helices appearing in some instance of \( \mathcal{M} \) and that are not in \( \mathcal{H}' \).

The simple occurrence problem consists in finding in the text \( K \) all positions where at least one possible instance of the multi-structure matches the putative helices \( K \). For example, considering the multi-structure \( \mathcal{M} = (\mathcal{H}, \mathcal{R}) \), the occurrence \( \{(1,3,4,5), \phi\} \), with \( \phi(1) = 1, \phi(3) = 3, \phi(4) = 5 \) and \( \phi(5) = 7 \), has no errors because it matches the instance \( \{1,3,5\} \). This applies typically to RNA sequences that are provided with a set of potential secondary structures, such as suboptimal secondary structures of Figure 2, or variants for bacterial RNase P RNA of Figure 3. When the multi-structure encodes a set of suboptimal predicted structures, this can serve for example to improve homology searching, even if the real structure is not know: Build the associated multi-structure, set of suboptimal predicted structures, this can serve for example to improve homology or variants for bacterial RNase P RNA of Figure 3. When the multi-structure encodes a set of potential secondary structures, such as suboptimal secondary structures of Figure 2, or variants for bacterial RNase P RNA of Figure 3. We define \( S(w,k,\ell) \) for the simple occurrence problem \( (h,k,\ell) \leftarrow (w) \in \mathcal{G} \}. \)

Equations for calculating \( S \) are given in Figure 4.
Searching for alternate RNA structures in genomic sequences

\[ S(\lambda, k, \ell) = 0 \]

\[ S(h_1 \circ h_2 \circ \ldots \circ h_q, k, \ell) = \]

Case 1: if \( K[k..\ell] \) is empty, then ||\( A[h_1 \circ \ldots \circ h_q] || \)

Case 2: otherwise, if \( \ell \sqsupset k \), then \( S(h_1 \circ \ldots \circ h_q, k+1, \ell) \)

Case 3: otherwise

\[
\begin{align*}
3-a: \text{Helix } k \text{ of the text is not in the multi-structure} \\
S(h_1 \circ \ldots \circ h_q, k+1, \ell) \\
3-b: \text{Helix } h_1 \text{ is not present in the text, but some helix of } A[h_1] \\
\min_{k \leq p \leq \ell} 1 + T(h_1, k, p) + S(h_2 \circ \ldots \circ h_q, \text{firstJuxt}(p), \ell) \\
3-c: \text{No helix of } A[h_1] \text{ is found in the text} \\
||A[h_1]|| + ||h_2 \circ \ldots \circ h_q, k, \ell|| \\
3-d: \text{Helix } h_1 \text{ is matched with helix } k \text{ of the text} \\
T(h_1, \text{firstNested}(k), \text{lastNested}(k)) + S(h_2 \circ \ldots \circ h_q, \text{firstJuxt}(k), \ell)
\end{align*}
\]

Fig. 4. Recurrence equations to compute \( S \) values for the simple occurrence problem. \( \lambda \) denotes the empty flat structure, \( k+1 \) the helix succeeding \( k \) for \( \sqsubset \), firstJuxt(\( k \)) the first helix according to \( \sqsubset \) that is juxtaposed with \( k \), firstNested(\( k \)) the first helix according to \( \sqsubset \) that is nested in \( k \), and lastNested(\( k \)) the last helix according to \( \sqsupset \) that is nested in \( k \). ||\( A[w] || + ||A[h_1 \circ \ldots \circ h_q]|| ||A[h_1]|| + ||A[h_2 \circ \ldots \circ h_q]||. Cases 1 and 2 are initial cases. We give below an example for case 3.
Property 1: The value $\min\{S(w,k_0,\ell_0), S \rightarrow w \in G\}$, where $k_0$ is the smallest helix of $K$ for the $\precsim$ ordering and $\ell_0$ the largest helix of $K$ for the $\subseteq$ ordering, gives the number of errors to match $\mathcal{M}$ against $K$, such as defined in the simple occurrence problem.

For the universal occurrence problem, analogous equations hold, where the two $\min$ operators in Equation (\circ) and in the caption of Figure 4 have to be replaced by $\max$ operators when computing $T$ and $||\mathcal{M}[w]|$. The number of errors to match $\mathcal{M}$ against $K$ is then given by $\max\{S(w,k_0,\ell_0), S \rightarrow w \in G\}$.

Initialization of all values in $T$ to $+\infty$

For each helix $\ell$ in the text $K$ in increasing order for $\precsim$

\[ w_{\ell,old} := \lambda \]

For each flat structure $w = h_1 \circ \ldots \circ h_q$ in $W$ in increasing order for $\precsim_{\text{lex}}$

\[ s := \text{length of the longest common suffix between } w \text{ and } w_{\ell,old} \]

For each helix $k$ in $K[1..q]$ in decreasing order for $\precsim$

For $i := s + 1 \text{ to } q$

compute $S'[i,k] := S(h_q \cdots h_{q-i} \circ \ldots \circ h_q,k,\ell)$ with Equations of Fig. 4 (\star)

End for $i$

For each helix $h$ in $\mathcal{H}$ such that $H \rightarrow h[w] \in G$

$T[h,k,\ell] := \min(T[h,k,\ell], S'[q - 1,k])$

End for $h$

End for $k$

$w_{\mathcal{M}} := w$

End for $w$

End for $\ell$

Fig. 5. Implementation of equations of Figure 4

It is possible to implement equations of Figure 4 using dynamic programming for $T$ and $S$. There are $mn^2$ values to compute for function $T$, where $m$ and $n$ are the numbers of helicities in $\mathcal{H}$ and $K$ respectively. Considering $S$, one can note that the first parameter $w$ is always a suffix of some flat structure. A suffix of $w = h_1 \circ \ldots \circ h_q$ is any flat structure of the form $h_i \circ \ldots \circ h_q$ for any $1 \leq i \leq q$. If we denote by $W$ the set of flat structures on $\mathcal{H}$ that appears in a right-hand-side of a production of $G$, the total number of all different suffixes of $W$ is bounded by $\sigma = \sum_{w \in W} |w|$. So the total number of different values to compute for $S$ and $T$ is $O(mn^2 + \sigma n^2)$. We still have to discuss the order of execution to compute $T$ and $S$. When computing $T(h,k,\ell)$, we need to access values of $S$ of the form $S(w,k,\ell)$, where $w$ is a flat structure nested in $h$. When computing $S(w,k,\ell)$, and supposing that all required values for $T$ are known, we only need to access values of $S$ that are of the form $S(w',k',\ell)$, where $w'$ is always a suffix of $w$. More precisely, either $w' = w$ and $k \preceq k'$, or $w'$ is a proper suffix of $w$ and $k \nsucceq k'$. Note also that the last parameter $\ell$ is unchanged. This means that to compute $S(w,k,\ell)$, we can forget all previously computed values for $S$ whose first parameter is not a suffix of $w$ and whose last parameter is not $\ell$. However, as some flat structures of $W$ may share common suffixes, some $S(w',k',\ell)$ values are used several times. To prevent redundant useless computations, it is thus necessary to order flat structures according to their suffixes. We
Searching for alternate RNA structures in genomic sequences

define the \( \sqsubseteq_{\text{lex}} \) order on \( W \) as the lexicographic ordering built on \( \sqsubseteq \), starting from the rightmost helices: \( h_1 \circ \cdots \circ h_{q-1} \circ h_q \sqsubseteq_{\text{lex}} h_1' \circ \cdots \circ h_{q'}' \circ h_{q-1}' \circ h_q' \) if, and only if, \( h_q \sqsubseteq h_q' \), or \( h_q = h_q' \) and \( h_1 \circ \cdots \circ h_{q-1} \sqsubseteq_{\text{lex}} h_1' \circ \cdots \circ h_{q-1}' \).

It follows from these remarks that helices \( \ell \) should be enumerated in increasing order for \( \sqsubseteq \), flat structures \( w \) of \( W \) in increasing order for the lexicographic ordering \( \sqsubseteq_{\text{lex}} \), helices \( k \) in decreasing order for \( \preceq \), and suffixes of a given flat structure in decreasing order for \( \preceq \).

Figure 5 shows the resulting algorithm, using a permanent two-dimensional table for \( T \) of size \( mn^2 \) and a temporary two-dimensional table \( S' \) for \( S \). For a given helix \( \ell \) and a flat structure \( w \), we eventually store in \( S'[j,k] \) the value of \( S \) for the suffix of \( w \) of size \( j \), compared to the text \( K[k..\ell] \), that is \( S'[j,k] = S(h_{q-(j-1)} \circ \cdots \circ h_q,k,\ell) \). The table \( S' \) is of size \( \gamma \times n \), where \( \gamma \leq m \) is the maximal length of a flat structure in \( W \), and \( n \) is the number of helices in \( K \). This algorithm thus requires space \( O(mn^2) \). As for the time complexity, the loop on \( \ell \) has \( O(n) \) iterations, and, for each \( \ell \) and \( w \), the loop on \( k \) has also \( O(n) \) iterations. In the worst case, there is no common suffixes between flat structures of \( W \): for each flat structure \( w \), the loop on \( i \) covers all helices of \( w \), in \( O(\gamma) \) executions of the \( (\ast) \) line. For any fixed \( \ell \) and \( k \), the total number of executions of the \( (\ast) \) line is thus bounded by \( \sigma \). Finally, each execution of the \( (\ast) \) line takes at worst \( O(n) \) time (loop on \( p \) with \( k < p \sqsubseteq \ell \), see case 3b of Figure 4). Taking all together, the algorithm runs in \( O(\sigma mn^3) \) worst-case time.

Note that the algorithm of Figure 5 computes the minimal number of errors. Retrieving the actual best alignments of the pattern with the text is done through backtracking in the \( T \) table, recomputing only the relevant \( S \) tables.

6 Conclusion

We have shown that our definition of RNA multi-structure can faithfully model several situations where alternate structures naturally arise. This result suggests many questions for future research: How to build automatically multi-structures? How to sample or enumerate them? How to compare them? How to search for structural elements in a multi-structure? In this paper, we have investigated the pattern matching problem, and have provided an algorithm to search for RNA multi-structures in a text. We have made the decision to stay at a abstract level. The RNA sequence and the text are both defined as sets of helices, that can be either nested or juxtaposed. However, there are several directions to make the model more realistic, such as adding compatibility relations or distance contraints between helices. This additional information also makes the algorithm significantly faster in practice. A prototype has been implemented and is available upon request.

Acknowledgments: The authors would like to thank Robert Giegerich (Bielefeld Universität) for fruitful discussions and valuable suggestions, which helped to improve significantly this work.

References


© A Saffarian, M Giraud and H Touzet
DOI: 10.15455/CMSR.2014.0002
Creative Commons BY-NC-ND 4.0 license
Saffarian, Giraud and Touzet

Ab initio Prediction of RNA Nucleotide Interactions with Backbone $k$-Tree Model

Liang Ding$^1$*, Xingran Xue$^1$, Sal LaMarca$^1$, Mohammad Mohebbi$^1$, Abdul Samad$^4$, Russell L. Malmberg$^{2,3}$, and Liming Cai$^{1,2}$*

$^1$Department of Computer Science, $^2$Institute of Bioinformatics and $^3$Department of Plant Biology, University of Georgia, GA 30602, USA, 
{lding, xrxue, slamarca, mohebbi}@uga.edu, russell@plantbio.uga.edu, 
cai@cs.uga.edu

$^4$Department of Computer Science, BUITEMS, Pakistan.
abdul.samad1@buitms.edu

Abstract. Given the importance of non-coding RNAs to cellular regulatory functions and rapid growth of RNA transcripts, computational prediction of RNA tertiary structure remains highly demanded yet significantly challenging. Even for a short RNA sequence, the space of tertiary conformations is immense; existing methods to identify native-like conformations mostly resort to random sampling to gain computational feasibility. However native conformations may not be examined and prediction accuracy may be compromised due to sampling. In particular, the state-of-the-art methods have yet to deliver the desired prediction performance for RNAs of length beyond 50.

This paper presents the work to tackle a key step in the RNA tertiary structure prediction problem, the prediction of the nucleotide interactions that constitute the desired tertiary structure. The research is established upon a novel graph model, called backbone $k$-tree, to markably constrain nucleotide interaction relationships in RNA tertiary structure. It is shown that the new model makes it possible to efficiently predict the optimal set of nucleotide interactions from the query sequence, including the interactions in all recently revealed families. Evident by the preliminary results, the new method can predict with a high accuracy the nucleotide interactions that constitute the tertiary structure of the query sequence, thus providing a viable solution towards ab initio prediction of RNA tertiary structure.

1 Introduction

In the past decade, there have been many revelations of the importance of non-coding RNAs to cellular regulatory functions and thus a growing interest in computational prediction of RNA tertiary structure [15], [17]. Nevertheless, RNA tertiary structure prediction from a single RNA sequence is a significant challenge. One major unresolved issue is in the immense space of tertiary conformations even for a short RNA sequence. Existing methods usually employ random sampling algorithms for computation feasibility, which assemble sampled tertiary motifs into native-like structures [6], [8], [12], [22], [25], [28]. To reduce the chance to miss native structures, the assembly

* To whom correspondence should be addressed.
Ding, Xue, LaMarca, Mohebbi, Samad, Malmberg and Cai

algorithms have mostly been guided with constraining structural models. For example, MC-Fold/MC-Sym [22] assumes the tertiary structure consists of 4-nt cyclic tertiary motifs constructible from the predicted secondary structure. Rosetta [6,7] de novo assembles tertiary structure from a database of 3-nt tertiary fragments. Other methods follow samplings that preserve the secondary structure [4, 25, 26] or intervention from human experts [13, 20]. However, these constraining models do not necessarily ensure that native conformations are examined. In particular, the state-of-the-art methods have yet to deliver the desired prediction accuracy for RNA sequences of lengths beyond 50 [15].

In this work, we introduce a novel method to predict nucleotide interactions from sequences as a key step toward accurate \textit{ab initio} prediction of tertiary structure. Accurate knowledge of the nucleotide interactions is crucial to predicting the tertiary structure of an RNA and subsequently predicting its functional roles. To predict nucleotide interactions, our method is guided by a novel graph model called a \textit{backbone k-tree}, for small integer $k$, to globally constrain the nucleotide interaction relationships (NIRs) that constitute the tertiary structure. In such a $k$-tree graph, nucleotides are organized into groups of size $k+1$, such that NIRs are permitted only for nucleotides belonging to the same group and groups are connected to each other with a tree topology (see section 2). This model was inspired by our recent discovery of the small treewidth of the NIR graphs for more than 3,500 RNA chains extracted from 1,984 resolved RNAs (Figure 1). We have been able to develop dynamic programming algorithms with $O(n^{k+1})$ time and space complexities, efficient for small $k$, to compute the optimal backbone $k$-tree spanning over the nucleotides on the query sequence, given a scoring function [9,10]

\begin{figure}[h]
\centering
\includegraphics[width=0.5\textwidth]{fig1.png}
\caption{Treewidth distribution of NIR graphs of more than 3,500 chains deriving from 1,984 resolved RNA tertiary structures in the RNA Structure Atlas [27]. The RNAs with treewidth larger than 18 are omitted due to their very small number. These treewidths are actually upper bounds computed by a program [5]; it is likely that the exact treewidths of the NIR graphs may actually be smaller.}
\end{figure}
Ab initio Prediction of RNA Nucleotide Interactions with Backbone $k$-Tree Model

To ensure that the computed optimal $k$-tree can actually yield the set of nucleotide interactions that constitutes the native tertiary structure, our method defines the scoring function over detailed patterns of nucleotide interactions within every group of $k + 1$ nucleotides. We consider nucleotide interactions from the established geometric nomenclatures [16] and nucleotide interaction families [18], [29], [31], including base-base, base-phosphate, and base-ribose as well as base-stacking interactions. To test our method, we adopted an improved 3-tree model and pre-computed candidates of interaction patterns for every group of 4 nucleotides, by searching through RNA Structure Atlas [27]; this contains annotated atom-level nucleotide interactions for nearly 3,000 resolved tertiary structures. We trained artificial neural networks (ANNs) to compute the confidence of every given nucleotide interaction and the confidence of every admissible nucleotide interaction pattern for every group of 4 given nucleotides. We filtered out unlikely interaction patterns and kept only those with high confidences. With this 3-tree model, our algorithm efficiently predicts an optimal set of nucleotide interactions from the query sequence within computational time $O(c^5 M n^3)$, where $M$ is a constant and $c \leq 20$ is the maximum number of candidate interaction patterns for one group of 4 nucleotides. We have implemented the algorithm into a program called BkTree, which may use known or predicted canonical (i.e., cis Watson-Crick) base pairs on the query sequence.

To evaluate our method for nucleotide interaction prediction, we tested BkTree on a benchmark set of 43 high resolution RNAs, which had been used to survey a number of state-of-the-art tertiary structure prediction methods [15]. The resolved, atom-level interactions were extracted with FR3D [27]. BkTree performed impressively well across the set of tested RNAs (Table 3), achieving the averaged sensitivity, PPV, and MCC values of 0.86, 0.78, and 0.82, respectively (discounting the input canonical base pairs). In comparison with previous programs MC [22], Rosetta [6], and NAST [12] that all assumed the secondary structure as a part of the input [15], it is clear that BkTree outperformed the other three programs in the MCC measure on this set of benchmark RNAs (Table 4, Figure 3). In particular, on the four representative RNAs that contain typical helices and junctions [15], BkTree gave the best performance on all but one RNA, for which BkTree acquired a higher sensitivity value but lower PPV than the MC program, resulting in a slightly lower MCC value (Table 5).

To evaluate the significance of our method to 3D conformation prediction, we used the program MC-Sym to model 3D conformations from the interactions predicted by BkTree and calculated RMSDs against the resolved structures. Since MC-Sym requires secondary structure for 3D conformation modeling, we identified 30 RNAs from the benchmark set for which their secondary structures are covered by the BkTree-predicted nucleotide interactions together with its input canonical base pairs. For the 4 representative RNAs listed in Table 5, BkTree outperforms MC and Rosetta on 3 of them.

2 Model and Methods

In this work, we consider all known types of nucleotide interactions of atomic-resolution [16], [18], [31]. In particular, with the base triangle model consisting of Watson-Crick (W), Hoogsteen (H), and sugar (S) edges, base-base interactions has been fully charac-
Ding, Xue, LaMarca, Mohebbi, Samad, Malmberg and Cai

terized into rich 12 geometric types and 18 interaction families [16], [18], according to involved edges, cis or trans, and parallel or anti-parallel, observed in crystal structures. For example the cWW family contains, in addition to the canonical (i.e., cis Watson-Crick) base pairs, many non-canonical base-base interactions through W edges. More recently, classifications of nucleotide interactions have been extended to base-backbone interactions. There are 10 families identified for base-phosphate interactions based on the position of the interacting hydrogen atom in the base [31]. Similarly, 9 additional families have been identified for base-ribose interactions [32]. A few base stacking interactions have also been classified. Table 1 summarizes these classes of nucleotide interactions, which also includes the backbone interaction between two neighboring nucleotides.

Table 1. Categories, types and families of RNA nucleotide interactions, mostly summarized from works [16], [18], [31,32]. It also includes the phosphodiester interaction between two neighboring nucleotides.

<table>
<thead>
<tr>
<th>Categories</th>
<th>Types (Interaction Families)</th>
<th>Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>Base pairs</td>
<td>cWW, tWW, cWH, tWH, cHW, tHW, cWS, tWS, cSW, tSW, cHH, tHH, cHS, tHS, cSH, tSH, cSS, tSS</td>
<td>18</td>
</tr>
<tr>
<td>Base-phosphates</td>
<td>0BPh, 1BPh, 2BPh, 3BPh, 4BPh, 5BPh, 6BPh, 7BPh, 8BPh, 9BPh</td>
<td></td>
</tr>
<tr>
<td>Base-riboses</td>
<td>0BR, 1BR, 2BR, 3BR, 4BR, 5BR, 6BR, 7BR, 9BR</td>
<td>9</td>
</tr>
<tr>
<td>Bases stackings</td>
<td>s35, s53, s33, s55</td>
<td>4</td>
</tr>
<tr>
<td>Backbone-backbone</td>
<td>phosphodiester</td>
<td>1</td>
</tr>
</tbody>
</table>

2.1 Backbone $k$-Tree Model

Let the query RNA sequence be $S = S_1S_2...S_n$, where $S_i \in \{A, C, G, U\}$, for $1 \leq i \leq n$. We denote an interaction between the $i$th and $j$th nucleotides, where $i < j$, with triple $\langle S_i(i), S_j(j), t \rangle$, for some interaction type $t$ shown in Table 1. Note that there are possibly two or more simultaneous interactions between the two nucleotides.

Given the native tertiary structure of the sequence $S$, we model the nucleotide interaction relationships (NIRs) within the tertiary structure with a graph $G = (V, E)$, where $V = \{S_i(i) : 1 \leq i \leq n\}$, such that $(S_i(i), S_j(j))$ is an edge in $E$ if and only if $i \neq j$ and $\langle S_i(i), S_j(j), t \rangle$ is an interaction for some $t$. We call $G$ the NIR graph of the sequence with the given structure. Because every two consecutive nucleotides are connected with the phosphodiester bond, every NIR graph of $n$ vertices contains all edges $\langle S_i(i), S_{i+1}(i+1) \rangle$, for $1 \leq i \leq n - 1$. These edges are called backbone edges.

In our recent investigation [9], we constructed NIR graphs for all RNAs whose tertiary structures were known from RNA Structure Atlas [26]. We discovered that an overwhelming majority of these RNAs are of small treewidths (Figure 1). Treewidth is a graph metric, which intuitively indicates how much a graph is tree-like. If a graph has treewidth bounded by $k$, any clique obtained by deleting vertices and edges and
Ab initio Prediction of RNA Nucleotide Interactions with Backbone $k$-Tree Model

contracting edges of the graph can contain at most $k + 1$ vertices [2]. Thus the distribution of treewidths suggest that NIRs in the RNA tertiary structures are in general not arbitrarily complex.

The concept of treewidth originated from the algorithmic graph theory. It is closely related to, and may be better explained with the notion of $k$-tree, which is central to this work.

**Definition 1.** [24] Let integer $k \geq 1$. The class of $k$-trees are graphs defined by the following inductive steps:

1. A $k$-tree of $k + 1$ vertices is a clique of $k + 1$ vertices;
2. A $k$-tree of $n$ vertices, for $n > k + 1$, is a graph consisting of a $k$-tree $G$ of $n - 1$ vertices and a vertex $v$, which does not occur in $G$, such that $v$ forms a $(k + 1)$-clique with some $k$-clique already in $G$.

Figure 2 shows a 3-tree with seven vertices in (a) and illustrates it in (b) with a tree-topology that connects the four 4-cliques in the graph.

**Fig. 2.** (a) 3-tree of 7 vertices by Definition 1, with the order of forming the four 4-cliques: with initial clique $\{1, 2, 3, 6\}$ (black edges), vertex 5 and blue edges added, then vertex 7 and red edges added, and finally vertex 4 and green edges added. (b) Illustration of the graph of (a) with a tree-topology connecting the four 4-cliques. (c) A backbone 3-tree for sequence AUUGGCA, of the same topology as shown in (a); backbone edges are in bold.

By [30], for any $k \geq 1$, a graph is of treewidth $\leq k$ if and only if it is a subgraph of a $k$-tree. Therefore, NIR graphs for an overwhelming majority of known RNA tertiary structures are constrained in topology by $k$-trees, for small values of $k$. Because technically, every graph of treewidth bounded by $k$ can be augmented with additional edges into a $k$-tree, we adopt such $k$-trees as the model for NIRs of the RNA tertiary structure.

**Definition 2.** Let $k \geq 1$ be an integer. The backbone $k$-tree for an RNA sequence is an augmented NIR graph of the sequence, which is a $k$-tree.

Figure 1(c) shows a backbone 3-tree for sequence AUUGGCA. Note that backbone $k$-trees differ from general $k$-trees in that a backbone $k$-tree has to the designated Hamiltonian path (consisting of all the backbone edges).

With the backbone $k$-tree model, in order to predict the set $I$ of nucleotide interactions from the query sequence, we propose to identify a backbone $k$-tree $G = (V, E)$ such that

$$(S_i^{(i)}, S_j^{(j)}) \in E \text{ if and only if } \exists t \langle S_i^{(i)}, S_j^{(j)}, t \rangle \in I$$
To ensure the identified $G$ actually corresponds to the set of interactions that constitute the native structure of the query sequence, we need to quantify nucleotide interactions for combinatorial optimization of such a backbone $k$-tree, as explained in the subsequent sections.

2.2 Quantification of Nucleotide Interactions

**Definition 3.** Let $q$ be a $(k+1)$-clique in a backbone $k$-tree of query sequence $S$. An interaction pattern (ip) for clique $q$ is a set $P_q$ of interactions for the nucleotides in $q$ such that for every interaction $\langle S_i^{(i)}, S_j^{(j)}, t \rangle$ in $P_q$, both nucleotides $S_i^{(i)}$ and $S_j^{(j)}$ are in clique $q$.

Given an $P_q$ for clique $q$, we define the induced subgraph by $P_q$, denoted with $B_q = \{q, E_{B_q}\}$ to be a subgraph of $q$ such that edge $(S_i^{(i)}, S_j^{(j)}) \in E_{B_q}$ only if interaction $\langle S_i^{(i)}, S_j^{(j)}, t \rangle \in P_q$ for some $t$.

**Definition 4.** Let $q$ be a $(k+1)$-clique in the in a backbone $k$-tree of query sequence $S$. The confidence of a given $P_q$ for clique $q$ is defined as

$$f(q,P_q,S) = \sum_{\langle S_i^{(i)}, S_j^{(j)}, t \rangle \in P_q} c_{q,B_q,t}$$

where $c_{q,B_q,t}$ is the confidence of interaction $\langle S_i^{(i)}, S_j^{(j)}, t \rangle$ given $q$ and subgraph $B_q$ induced by $P_q$.

In the Section 3, we will introduce artificial neural networks (ANNs) to compute confidence $c_{q,B_q}^{(i,j)}$.

For every clique $q$, with $\mathcal{D}(q)$, we denote the finite set of all ips for $q$. In the practical application, we may only include those ips in $\mathcal{D}(q)$ which have “high” confidences (e.g., above certain threshold). Let $I$ be a set of interactions. By notation $I_{|q}$, we mean the maximal size subset of $I$ that is an ip for $q$.

**Definition 5.** Let $k$ be any fixed integer $\geq 2$. The nucleotide interaction prediction problem NIP($k$) is, given an input query sequence $S$, to identify a backbone $k$-tree $G^* = (V,E^*)$ as well as a set $I^*$ of nucleotide interactions that constitutes the tertiary structure of $S$, such that every interaction $\langle S_i^{(i)}, S_j^{(j)}, t \rangle \in I^*$ implies edge $(S_i^{(i)}, S_j^{(j)}) \in E^*$ and

$$I^* = \arg \max_{I, G} \left\{ \sum_{\langle q,I\rangle \in \mathcal{D}(q)} f(q,I_{|q},S) \right\}$$

2.3 Overview of the Method

Our method consists of three major components to solve the NIP($k$) problem, for any fixed $k \geq 2$. The first component is data repositories including NIPDB and NIPCCTable. NIPDB is a database of all possible interaction patterns (ips) for every $(k+1)$-clique,
Ab initio Prediction of RNA Nucleotide Interactions with Backbone k-Tree Model

which was established by searching through the RNA Structure Atlas [27]. For every such clique, its ips in NIPDB are extracted and ranked when the query sequence is pre-processed. NIPCTable is a matrix for compatibility between every pair of ips for two cliques that share all but one nucleotide. The compatibility is checked by the dynamic programming algorithm computing the NIP(k) problem.

The second component is a set of artificial neural networks (ANNs) to compute confidence for any given interaction type \( t \) between any two given nucleotides \( S^{(i)}_h \) and \( S^{(j)}_h \) on the query sequence. The computed confidences for interactions are then used to compute confidence of an ip for every \( (k+1) \)-clique, as formulated in equation (1). For every such clique \( q \), all ips of \( q \) obtained from database NIPDB are ranked according to their confidence values. Often the number of ips with significant confidence values is small, e.g., \( \leq 20 \); ips of significant scores are included as ip candidates into the set \( \mathcal{D}(q) \) for \( q \). The detailed construction of the ANNs will be described in the next section.

The third component is a dynamic programming algorithm solving the NIP(k) problem, using the prepared data and preprocessing results from the first two components. From the input query sequence, the algorithm produces a backbone k-tree \( G^* \) as well as a set \( I^* \) of nucleotide interactions, maximizing the aggregate confidence value across all \( (k+1) \)-cliques in \( G^* \) (see equations (2) and (1)). The relationship between \( G^* \) and \( I^* \) is that, for every \( (k+1) \)-clique \( q \) in the k-tree \( G^* \), there is a maximal subset of nucleotide interactions \( I_q \subseteq I^* \) being an ip for \( q \), such that \( I^* = \bigcup_q I_q \). The next section describes the details of the dynamic programming algorithm.

3 Algorithms

3.1 ANNs for Computing Interaction Confidence

Let the query sequence \( S = S_1S_2\ldots S_n \) of \( n \) nucleotides, where \( S_i \in \{A,C,G,U\} \), for \( 1 \leq i \leq n \). Technically we considered all \( (k+1) \)-cliques formed by \( k+1 \) vertices \( \{S^{(h)}_{h_0}, S^{(h_1)}_{h_1}, \ldots, S^{(h_k)}_{h_k}\} \), where \( 1 \leq h_0 < h_1 < \cdots < h_k \leq n \). Let \( q = (V,E) \) be such a clique and \( B_q = (V, E_{B_q}) \), where \( E_{B_q} \subseteq E \), be any subgraph of \( q \). For every edge \( (S^{(i)}_h, S^{(j)}_h) \in E_{B_q} \), and every possible interaction \( (S^{(i)}_h, S^{(j)}_h, t) \) of type \( t \), we constructed an ANN \( \mathcal{A}_{q,B_q,t}^{(i,j)} \) to calculate confidence \( c_{q,B_q,t}^{(i,j)} \) that interaction \( (S^{(i)}_h, S^{(j)}_h, t) \) occurs in the subgraph \( B_q \) of clique \( q \).

Each ANN \( \mathcal{A}_{q,B_q,t}^{(i,j)} \) consists of an input layer, a hidden layer, and an output layer. The output layer is a single unit depicting a confidence value for interaction \( (S^{(i)}_h, S^{(j)}_h, t) \). The input layer consists of input units representing the selected global and local features shown in Table 2. The features included the sequence length and the distance between the involved nucleotides as well as neighboring nucleotide types. In addition, we included the information of assumed canonical base pairs\(^1\) within the query sequence. The complete list of features selected for the trainings are given in the Table 2.

\(^1\) These are known or predicted Watson-Crick and wobble base pairs. Note that they do not necessarily constitute all information about the secondary structure.
Table 2. Features selected from a given \((k+1)\)-clique \(q\) and given subgraph \(B_q\) of \(q\) for training ANN \(\mathcal{N}_q^{(i,j)}\). CBP is an abbreviation for canonical base pair. A Component (Cp) is defined as the maximal subsequence consisting of two or more nucleotides each involved in a CBP.

<table>
<thead>
<tr>
<th>Feature</th>
<th>Value</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Seq. length</td>
<td>An integer</td>
<td>Length of a training sequence containing (q).</td>
</tr>
<tr>
<td>Distances</td>
<td>(k) integers</td>
<td>Distances between every two nucleotides in the sequential order in (q).</td>
</tr>
<tr>
<td>Number of Cps</td>
<td>(k) integers</td>
<td>Number (one of ({0,1,2,3,-1})) of Cps on the subsequence between every two nucleotides in the sequential order. 3 means there are at least 3 Cps; -1 means the two nucleotides are neighboring nucleotides on the sequence.</td>
</tr>
<tr>
<td>Neighbor nts.</td>
<td>(k+1) 4-mers</td>
<td>One 4-mer (of letter (A, C, G, U)) for every nucleotide in (q), where the first two letters and the last two letter of the 4-mer indicate the two nts to the left and to the right of the nucleotide, respectively, and letter (N) is used when there is no neighbor.</td>
</tr>
<tr>
<td>Neighbor CBPs</td>
<td>(k+1) 4-mers</td>
<td>One 4-mer (of binary bits) for every nucleotide in (q), where the first two bits and the last two bits of the 4-mer indicate the two nts to the left and to the right of the nucleotide are involved in CBPs, respectively, and letter (N) is used when there is no neighbor.</td>
</tr>
<tr>
<td>Edge properties</td>
<td>up to (\binom{k+1}{2}) integers</td>
<td>For every edge in the subgraph (B_q) of (q), value 0 indicates both nts are involved in a CBP; -1 (resp. +1) indicates exclusively left (resp. right) nt is involved in a CBP; 2 indicates either is near a CBP; and -2 indicates both are far away (distant beyond 3 nts) from a CBP.</td>
</tr>
</tbody>
</table>

We adopted conventional methods to construct and train the ANNs [21], typically the technique of back-propagation with gradient descent, using a fixed-size network. This is based on the calculation of the error by taking the first derivatives of half the Euclidean distance between the output and target and back-propagating it towards the input layer, over the whole training set. Each weight is then updated according to the error contribution of each unit, the error of each output unit and a learning rate. The logistic sigmoid was used as the activation functions for each unit. The updating is repeated until the training error converges to a minimum or the cross-validation error starts to rise, due to over-fitting. The learning rate 0.03 was the value that yielded the best results for a subset of 895 RNAs from RNA Structure Atlas.

The trained ANNs can be applied to compute confidence for interaction patterns. In particular, given a \((k+1)\)-clique \(q = \{S_{h_1}^{(i)}, S_{h_2}^{(j)}, \ldots, S_{h_{k+1}}^{(j)}\}, 1 \leq h_1 < \cdots < h_{k+1} \leq n\), let \(P_q\) be an ip for \(q\) and let \(B_{P_q}\) be the underlying graph for \(P_q\) which is a subgraph of clique \(q\). Then the trained ANN \(\mathcal{N}_q^{(i,j)}\) can be applied on each edge \((S_i^{(i)}, S_j^{(j)}) \in E_{B_{P_q}}\) and each type \(t\) to compute the confidence score \(c_{q,B_{P_q},t}^{(i,j)}\) for interaction \((S_i^{(i)}, S_j^{(j)}, t)\). The confidence \(f(q,P_q,S)\) of \(P_q\) for \(q\) is computed with the equation (1).

32
**Ab initio** Prediction of RNA Nucleotide Interactions with Backbone k-Tree Model

Then for $q$, all the ips $P_q$’s are ranked according to their confidences $f(q, P_q, S)$, and only significant top $m$ ips are included in the candidate set $\mathcal{Q}(q)$. We have chosen $m \leq 20$ in the performance evaluations as our experiments results had showed that a larger $m$ could not help to improve the results.

### 3.2 Algorithm for NIP($k$) problem

Roughly speaking, the algorithm for NIP($k$) problem considers every ($k+1$)-clique, from which recursive creations of more cliques are all examined. For every newly created clique $q$, all ips from $\mathcal{Q}(q)$ are considered but eventually exactly one of them is chosen for $q$. The algorithm follows the basic process of creating $k$-tree given in Definition 1. However, because the identified $k$-tree is a backbone $k$-tree that contains all backbone edges, the process is not straightforward. We need the following notations for an introduction to the algorithmic idea. By interval $[i..j]$, for $i \leq j$, we mean the set of consecutive integers between $i$ and $j$, inclusive. Two intervals $[i..j]$ and $[h..l]$ are non-overlapping if either $j \leq h$ or $l \leq i$. Formally, let the query sequence be $S = S_1S_2\ldots S_n$ and $q$ be a clique formed by $k+1$ vertices $\{S_{b_1}, S_{b_2}, \ldots, S_{b_{k+1}}\}$, where $1 = h_0 \leq h_1 < h_2 < \cdots < h_{k+1} \leq n = h_{k+2}$. Let $A$ be a set of non-overlapping intervals and $P_q \in \mathcal{Q}(q)$ be an ip for clique $q$.

We define function $M(q, A, P_q, S)$ to be the maximum confidence of a $k$-tree constructed beginning from clique $q$, which includes all backbone edge $\{S_{b_1}^{(i)}, S_{b_{k+1}}^{(i+1)}\}$ for integers $i$ and $i+1$ both contained in the same interval in $A$. Then we obtain the following recurrence:

$$
M(q, A, P_q, S) = \max_{\substack{p \in \mathcal{Q}(P_q) \\ p \neq q}} \max_{\substack{S_{b} \notin q, \\ y \in A \cap [i..j] \in A}} \{M(p, B, P_p, S) + M(q, C, P_q, S) + f(q, P_q, S)\} \tag{3}
$$

where abbreviations $q[i..j] = q \cup \{S_{b}^{(i)}\} \setminus \{S_{b}^{(j)}\}$, $\mathcal{Q}(P_q, P_p)$ asserts that the chosen ip $P_p$ be compatible with $P_q$, and $\mathcal{Q}(B, C)$ represents the choices of two sets of intervals, $B$ and $C$, which satisfy constraints

- (a) $\{[i..j], [y..j]\} \subseteq B$, $\{[w..x], [x..z]\} \subseteq C$, for applicable $w$ and $z$; and
- (b) $B \cup C = A \cup \{[i..j], [y..j]\} \setminus \{[i..j]\}$, and $B \cap C = \emptyset$.

Recurrence (3) gives an iterative process to produce a backbone $k$-tree. The intuitive idea is to create a new clique $p$ from $q$ by introducing a new nucleotide vertex $S_{b}^{(i)}$ to the partially constructed $k$-tree. This results in possibly two or more sub-$k$-trees, one starting from $p$ and the others from $q$ (but not including $S_{b}^{(j)}$). Since the two or more sub-$k$-trees will never join together again, interval sets are used to ensure backbone edges will be properly created. Essentially, the constructed $k$-tree corresponding to the value of function $M(q, A, P_q, S)$ contains only those backbone edges that connect the nucleotides of indexes specified in the intervals in $A$. In particular, starting from clique
Ding, Xue, LaMarca, Mohebbi, Samad, Malmberg and Cai

$q$ of $k + 1$ vertices \{\(S^{(h_1)}\), \(S^{(h_2)}\), \ldots, \(S^{(h_{k+1})}\)\}, to compute a backbone $k$-tree that contains all the backbone edges, we need to set $A = \{[h_i, h_{i+1}] : 0 \leq i \leq k + 1\}$, where $h_0 = 1$ and $h_{k+2} = n$.

The confidence score of the produced $k$-tree is computed as the sum of confidence scores of ips chosen for all involved $(k + 1)$-cliques. The chosen ips need to be compatible across the cliques when they share nucleotide interactions or even just nucleotides. This is ensured by the assertion $\mathcal{P}(P_q, P_p)$, which checks (1) $P_q$ and $P_p$ have the same set of interactions on the edges shared by cliques $q$ and $p$ by looking up table NIPCCTable; and (2) any pattern of interactions between a single nucleotide and multiple others has to exist in the structure database.

To complete the recurrence, we need the following base case:

\[ M(q, A, P_q, S) = 0 \quad \text{if} \quad A = \emptyset \]

To identify the desired backbone $k$-tree $G^*$, we maximize $M(q, A, P_q, S)$ over all starting clique $q$ and all ip $P_q \in \mathcal{D}(q)$. The associated set $I^*$ of nucleotides is just the union of the chosen ips for all $(k + 1)$-cliques in $G^*$.

Recurrence (3) naturally offers a dynamic programming solution. Function $M(q, P_q, A, S)$ can be computed by establishing a table with dimensions for $q$, $P_q$, and $A$. With the base cases, the table is computed bottom-up, from $A = \emptyset$, using the recurrence (3).

### 3.3 Improved Algorithms

Simply implementing the above outlined algorithm would require $O(n^{k+1})$ memory space and $O(n^{k+2})$ computation time for every fixed value of $k$. Following the same idea but creating $(k + 1)$-cliques from $k$-cliques instead leads to an improved dynamic programming algorithm to solve the NIP($k$) problem, with a little more sophisticated steps to navigate through $k$-cliques. The improved algorithm uses $O(n^k)$ amount of memory space and $O(n^{k+1})$ amount of time for every fixed value of $k$ [9, 10].

The efficiency can be further improved by demanding that every $(k + 1)$-clique in backbone $k$-trees contains two consecutive nucleotides $S^{(j)}_i$ and $S^{(j+1)}_i$ for some $i$. That is, every interaction pattern for a $(k + 1)$-clique always contains at least one backbone edge. This allows a further reduction of computation time to $O(n^k)$. Testing on the case $k = 3$ has shown that the constrained backbone 3-tree model maintains the similar capability to account for sophisticated nucleotide interactions as the “standard” backbone 3-tree model. In addition the constraint may enforce the construction of the 3-tree to follow backbone edges, providing more controls on the 3-tree construction. Finally, the constraint also significantly reduced the number of cases that the ANNs need to consider in their construction.

### 3.4 Implementation

The NIPDB database construction was implemented by Python, where Prody package [3] was adopted to search RNA Structure Atlas. Afterward, NIPCCTable, the matrix for ip consistence and compatibility was developed using Python. Training and building of ANNs were realized with WEKA package [19]. Finally, confidences of ips admissible
Ab initio Prediction of RNA Nucleotide Interactions with Backbone \( k \)-Tree Model

for every clique \((k+1)\)-clique in the query sequence was computed by programs in Python.

We implemented in C++ the dynamic programming algorithm into a program called BkTree. We ran the evaluation tests on a Red Hat 4.8.2-7 server with 4 Intel Quad core X5550 Xeon Processors, 2.66GHz 8M Cache and 70GB Memory.

4 Performance Evaluation

4.1 Test Data

We implemented our method in the program BkTree. We evaluated our method through testing BkTree on a list of 43 RNAs of high resolution structure data, which had been used as a benchmark set to evaluate a number of state-of-the-art tertiary structure prediction methods in the survey [15]. 18 of the RNA sequences are of length \( \geq 50 \). In developing the ANNs for computing interaction confidences, 7 of these RNAs were not included in the training data.

Given the recent progress made in RNA secondary structure prediction [15], [26], we believe that canonical base pairs may be routinely predicted with a fair accuracy. Therefore, we have allowed the program BkTree to accept known or predicted canonical base pairs along with the query sequence as input. Note that the knowledge of canonical base pairs does not necessarily imply the whole secondary structure, which is often a part of input to most of the existing RNA 3D prediction methods. In our test, we extracted canonical base pairs of a RNA from FR3D analyzed interactions [27].

4.2 Overall Performance

We evaluated the quality of the predicted nucleotide interactions by the sensitivity (STY) and positive predictive value (PPV) against the FR3D-analyzed interactions [27]. In order to take into account the effects of both true positive and false positive rates in one measure, the Matthews correlation coefficient (MCC), defined in [15] as

\[
\text{MCC} := \sqrt{\frac{\text{PPV} \times \text{STY}}{1 - \text{PPV} - \text{STY}}},
\]

was also calculated.

Table 3 summarizes the overall performance of BkTree on the benchmark set. On a large majority of RNAs, the sensitivity is decently high. Note that the STY and PPV calculations excluded the canonical base pairs. The sensitivity result indicates that our method has a high accuracy in identifying non-canonical interactions that may be crucial to tertiary structures. This is true even for those longer RNAs. We further note that for the 7 RNAs that were not included in the training data, BkTree also performed extremely well.

4.3 Performance Comparison with Other Methods

We compared our program BkTree with the programs MC, Rosetta, and NAST on the capability to predict nucleotide interactions. These other methods had been surveyed
and evaluated in [15] based on their ability to identify both base pairing and base stacking interactions. We removed base-phosphate and base-ribose interactions from our prediction results. We incorporated the canonical base pairs into our results because these other methods include all interactions from the input secondary structure.

Figure 3 shows the MCC curves for MC, Rosetta, NAST, and BkTree on the benchmark set of RNAs. Data of RNAs failed by a program were not included in the calculation. We note that for every RNA, these other programs produced more than one conformation so the results were averaged for these comparisons. The figure demonstrates that BkTree overall outperformed the other three programs in predicting non-canonical base pairing and base stacking interactions.

Table 4 gives comparisons on average performance between the four methods. In general, Bktree produced much better average results than Rosetta and NAST, and comparable average results with MC, for which BkTree shows better average STY value than MC, whereas MC gives better average PPV. On MCC values, BkTree had an edge over MC. On RNAs of length ≥ 50, BkTree maintained almost the same average MCC as it did on the whole set.
Ab initio Prediction of RNA Nucleotide Interactions with Backbone \(k\)-Tree Model

4.4 Significance to 3D conformation prediction

To evaluate the significance of our method to 3D conformation prediction, we used MC-Sym [22] to model 3D conformations from the interactions predicted by BkTree and calculated RMSDs against the resolved structures. We note that MC-Sym does not accept interactions of categories other than base-pair and base stacking; the correctly predicted base-phosphate and base-ribose interactions by our methods were discarded by MC-Sym to produce 3D folds. The deviation index (DI) [23], a measure that accounts for both RMSD and MCC, defined as the quotient of them, was also calculated. Table 5 presents the performance values on the 4 representative RNAs chosen in [15] which typically contain two hairpins and two junctions. Since both MC and Rosetta allow prediction of multiple optimal or suboptimal folds, we chose the averaged values of their solutions. We note that to model 3D conformations with MC using our predicted interaction data, we needed the secondary structure of the tested RNA to be covered by the input canonical base pairs together with the interactions predicted by BkTree. RNA 2QUS failed on this requirement. The averaged RMSDs achieved by BkTree for the rest 3 RNAs are significantly smaller than those achieved by MC and Rosetta.

5 Discussion and Conclusion

Our method is the first \textit{ab initio} predict RNA non-canonical interactions of all types. Evaluation of the results have highlighted its potential as an important step toward accurate \textit{ab initio} 3D structure prediction. We attribute the encouraging preliminary results to the recent growth of knowledge in high-resolution nucleotide interaction data as well as to the novel backbone \(k\)-tree modeling of nucleotide interaction relationships. The latter makes it possible to markedly reduce the space of solutions for the nucleotide interaction prediction problem to one that can be feasibly searched in polynomial time.

Our method differs from others also in its direct prediction of nucleotide interactions whereas the others mostly attempt 3D conformation construction before producing nucleotide interactions. The difference makes it difficult to compare their performances, especially when a 3D structure is not the direct output of a software, e.g., RNA-MoIP [26]. Therefore, the MCC comparison with MC was probably more appropriate than the comparison with RNA-MoIP, since the results of MC were based on interactions from the RNA Structure Atlas and so did BkTree, while RNA-MoIP used Interaction Network Fidelity [11] in calculating the MCC values. The contrast is more evident when using MC-Sym to model 3D conformations from interaction data predicted by BkTree. Even though the predicted base-phosphate and base-ribose interactions have to be discarded, the resulted RMSDs seem to correlate with the MCC values (Table 5).

The evaluation tests have also revealed some issues with BkTree. First, the complexity of structures has an impact on our prediction results. Typically, BkTree underperformed on some of the RNAs with pseudoknots or 4-way junctions. Table 5 shows that BkTree loses to MC on MCC value for only one representative RNA 2QUS, which contains a pseudoknot. The underperformance is likely due to the 3-tree model that is a little too weak for complex structures. For example, the best 3-tree can include at most 83 interactions out of total 95 interactions of tRNA 2D3U, indicating a higher treewidth is needed for the NIR graph of this RNA. To improve prediction performance for such
RNAs, an algorithm may need to be based on the backbone 4-tree model. Our method is not ineffective for handling multi-way junctions or pseudoknots, e.g., RNA 2GIS in Table 3. Fixing a specific \( k \)-tree model, it is the NIR graph treewidth of an RNA that determines the performance on the RNA.

Second, the NIR graph treewidth is also related to scalability of our method. The current algorithm for the nucleotide prediction problem has the complexity \( O(n^3) \) for both time and memory requirements. With a large hidden constant in the polynomial, the implemented program BkTree typically runs in 2 to 3 hours on an RNA of length 100 and uses several Gigabytes of memory. This is because the current prototype has aimed at accuracy without optimization in computational efficiency. However, the problem (based on the \( k \)-tree model) has an inherent complexity of \( O(n^k) \); our method is scalable to suit longer and more complex RNAs, e.g., which require the 4-tree model.

Third, due to the lack of tools to model 3D conformations from nucleotide interactions of all types, it is an immediate future task of ours is to develop such a tool that can be pipelined with a program like BkTree for \textit{ab initio} 3D structure prediction. We perceive such a task to be feasible. This is because the output of program BkTree contains not only the predicted nucleotide interactions but also a backbone 3-tree that decomposes nucleotides according to their interconnectivity. The given 3-tree can be the basis for very efficient algorithms for computing a desirable optimization function on 3D conformations [1].

Acknowledgments

We thank Christian Laing for the provided raw data used in the survey paper [15]. This work was supported in part by NSF IIS grant (award No: 0916250).

References

Ab initio Prediction of RNA Nucleotide Interactions with Backbone k-Tree Model


**Ab initio** Prediction of RNA Nucleotide Interactions with Backbone \(k\)-Tree Model

Table 3. Nucleotide interaction prediction results by BkTree on the benchmark set used in the survey [15]. The number of canonical base pairs (CPBs) and number of non-canonical interactions (NCIs) are listed. The sensitivity (STY), PPV and MCC were calculated, excluding the canonical bases pairs used as a part of the input. The data of the 7 RNAs not used for training ANNs are displayed with the bold font.

<table>
<thead>
<tr>
<th>PDB ID</th>
<th>Length</th>
<th># CBPs</th>
<th># NCIs</th>
<th>STY</th>
<th>PPV</th>
<th>MCC</th>
<th>Structure complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>2F8K</td>
<td>16</td>
<td>6</td>
<td>14</td>
<td>85</td>
<td>85</td>
<td>0.8571</td>
<td>Hairpin</td>
</tr>
<tr>
<td>2AB4</td>
<td>20</td>
<td>6</td>
<td>20</td>
<td>100</td>
<td>90</td>
<td>0.9534</td>
<td>Hairpin</td>
</tr>
<tr>
<td>361D</td>
<td>20</td>
<td>5</td>
<td>17</td>
<td>70</td>
<td>57</td>
<td>0.6351</td>
<td>Hairpin</td>
</tr>
<tr>
<td>2ANN</td>
<td>23</td>
<td>3</td>
<td>24</td>
<td>75</td>
<td>66</td>
<td>0.7071</td>
<td>Hairpin</td>
</tr>
<tr>
<td>IRLG</td>
<td>25</td>
<td>5</td>
<td>22</td>
<td>95</td>
<td>63</td>
<td>0.7793</td>
<td>Hairpin, internal loop</td>
</tr>
<tr>
<td>2QUX</td>
<td>25</td>
<td>9</td>
<td>22</td>
<td>90</td>
<td>71</td>
<td>0.8058</td>
<td>Hairpin</td>
</tr>
<tr>
<td>387D</td>
<td>26</td>
<td>4</td>
<td>23</td>
<td>86</td>
<td>68</td>
<td>0.7744</td>
<td>Hairpin</td>
</tr>
<tr>
<td>1MSY</td>
<td>27</td>
<td>6</td>
<td>39</td>
<td>97</td>
<td>92</td>
<td>0.9502</td>
<td>Hairpin</td>
</tr>
<tr>
<td>IL2X</td>
<td>28</td>
<td>8</td>
<td>34</td>
<td>88</td>
<td>88</td>
<td>0.8823</td>
<td>Pseudoknot</td>
</tr>
<tr>
<td>2AP5</td>
<td>28</td>
<td>8</td>
<td>29</td>
<td>82</td>
<td>66</td>
<td>0.7427</td>
<td>Pseudoknot</td>
</tr>
<tr>
<td>IJID</td>
<td>29</td>
<td>8</td>
<td>31</td>
<td>93</td>
<td>72</td>
<td>0.8235</td>
<td>Hairpin, internal loop</td>
</tr>
<tr>
<td>1OOA</td>
<td>29</td>
<td>8</td>
<td>29</td>
<td>93</td>
<td>72</td>
<td>0.8242</td>
<td>Hairpin, internal loop</td>
</tr>
<tr>
<td>430D</td>
<td>29</td>
<td>6</td>
<td>37</td>
<td>94</td>
<td>77</td>
<td>0.8577</td>
<td>Hairpin, internal loop</td>
</tr>
<tr>
<td>3SNP</td>
<td>30</td>
<td>12</td>
<td>31</td>
<td>93</td>
<td>85</td>
<td>0.8932</td>
<td>Hairpin, internal loop</td>
</tr>
<tr>
<td>2OZB</td>
<td>33</td>
<td>10</td>
<td>33</td>
<td>93</td>
<td>79</td>
<td>0.8641</td>
<td>Hairpin, internal loop</td>
</tr>
<tr>
<td>1MJ1</td>
<td>34</td>
<td>10</td>
<td>44</td>
<td>84</td>
<td>84</td>
<td>0.8409</td>
<td>Hairpin, internal loop</td>
</tr>
<tr>
<td>1ET4</td>
<td>35</td>
<td>8</td>
<td>40</td>
<td>67</td>
<td>84</td>
<td>0.7546</td>
<td>Hairpin, internal loop</td>
</tr>
<tr>
<td>2HW8</td>
<td>36</td>
<td>12</td>
<td>44</td>
<td>93</td>
<td>80</td>
<td>0.8655</td>
<td>Hairpin, internal loop</td>
</tr>
<tr>
<td>II6U</td>
<td>37</td>
<td>15</td>
<td>47</td>
<td>91</td>
<td>89</td>
<td>0.9053</td>
<td>Hairpin, internal loop</td>
</tr>
<tr>
<td>IFFT</td>
<td>38</td>
<td>10</td>
<td>38</td>
<td>81</td>
<td>63</td>
<td>0.7184</td>
<td>Hairpin, internal loop</td>
</tr>
<tr>
<td>1ZHO</td>
<td>38</td>
<td>13</td>
<td>46</td>
<td>95</td>
<td>83</td>
<td>0.8911</td>
<td>Hairpin, internal loop</td>
</tr>
<tr>
<td>1SO3</td>
<td>47</td>
<td>18</td>
<td>53</td>
<td>88</td>
<td>79</td>
<td>0.8404</td>
<td>Hairpin, internal loop</td>
</tr>
<tr>
<td>1XJR</td>
<td>47</td>
<td>15</td>
<td>55</td>
<td>83</td>
<td>80</td>
<td>0.8215</td>
<td>Hairpin, internal loop</td>
</tr>
<tr>
<td>1U63</td>
<td>49</td>
<td>17</td>
<td>50</td>
<td>94</td>
<td>65</td>
<td>0.7833</td>
<td>Hairpin, internal loop</td>
</tr>
<tr>
<td>2PXK</td>
<td>49</td>
<td>16</td>
<td>66</td>
<td>98</td>
<td>94</td>
<td>0.9632</td>
<td>Hairpin, internal loop</td>
</tr>
<tr>
<td>2FK6</td>
<td>53</td>
<td>20</td>
<td>58</td>
<td>77</td>
<td>70</td>
<td>0.7385</td>
<td>Pseudoknot, 3-way junction</td>
</tr>
<tr>
<td>3ESC</td>
<td>53</td>
<td>21</td>
<td>65</td>
<td>84</td>
<td>73</td>
<td>0.7877</td>
<td>3-way junction (riboswitch)</td>
</tr>
<tr>
<td>1MZP</td>
<td>55</td>
<td>17</td>
<td>73</td>
<td>64</td>
<td>73</td>
<td>0.6876</td>
<td>Hairpin internal</td>
</tr>
<tr>
<td>1DK1</td>
<td>57</td>
<td>24</td>
<td>65</td>
<td>100</td>
<td>89</td>
<td>0.9436</td>
<td>3-way junction</td>
</tr>
<tr>
<td>1MM5</td>
<td>58</td>
<td>20</td>
<td>86</td>
<td>74</td>
<td>82</td>
<td>0.7814</td>
<td>3-way junction</td>
</tr>
<tr>
<td>3EGZ</td>
<td>65</td>
<td>23</td>
<td>72</td>
<td>70</td>
<td>66</td>
<td>0.6849</td>
<td>3-way junction (riboswitch)</td>
</tr>
<tr>
<td>2QUX</td>
<td>69</td>
<td>26</td>
<td>81</td>
<td>75</td>
<td>76</td>
<td>0.7577</td>
<td>Pseudoknot, 3-way junction</td>
</tr>
<tr>
<td>1KKK</td>
<td>70</td>
<td>28</td>
<td>87</td>
<td>96</td>
<td>92</td>
<td>0.9440</td>
<td>Hairpin, internal loop</td>
</tr>
<tr>
<td>2DU3</td>
<td>71</td>
<td>27</td>
<td>75</td>
<td>78</td>
<td>70</td>
<td>0.7433</td>
<td>4-way junction (tRNA)</td>
</tr>
<tr>
<td>2O1U</td>
<td>71</td>
<td>29</td>
<td>84</td>
<td>90</td>
<td>83</td>
<td>0.8692</td>
<td>3-way junction (riboswitch)</td>
</tr>
<tr>
<td>1SJ4</td>
<td>73</td>
<td>19</td>
<td>83</td>
<td>78</td>
<td>81</td>
<td>0.7976</td>
<td>Pseudoknot, 4-way junction</td>
</tr>
<tr>
<td>1PSO</td>
<td>77</td>
<td>29</td>
<td>86</td>
<td>97</td>
<td>77</td>
<td>0.8716</td>
<td>Hairpin, internal loop</td>
</tr>
<tr>
<td>3D2G</td>
<td>77</td>
<td>28</td>
<td>103</td>
<td>80</td>
<td>88</td>
<td>0.8435</td>
<td>3-way junction (riboswitch)</td>
</tr>
<tr>
<td>2H0J</td>
<td>79</td>
<td>27</td>
<td>100</td>
<td>87</td>
<td>84</td>
<td>0.8572</td>
<td>3-way junction (riboswitch)</td>
</tr>
<tr>
<td>2GD1</td>
<td>80</td>
<td>32</td>
<td>100</td>
<td>84</td>
<td>80</td>
<td>0.8197</td>
<td>3-way junction (riboswitch)</td>
</tr>
<tr>
<td>2GIS</td>
<td>94</td>
<td>36</td>
<td>125</td>
<td>87</td>
<td>82</td>
<td>0.8485</td>
<td>Pseudoknot, 4-way junction (riboswitch)</td>
</tr>
<tr>
<td>1LNQ</td>
<td>97</td>
<td>38</td>
<td>124</td>
<td>85</td>
<td>79</td>
<td>0.8254</td>
<td>3-way junction (SRP)</td>
</tr>
<tr>
<td>1MFQ</td>
<td>128</td>
<td>49</td>
<td>164</td>
<td>81</td>
<td>76</td>
<td>0.7895</td>
<td>3-way junction (SRP)</td>
</tr>
</tbody>
</table>

Proceedings of the ECCB’14 workshop on Computational Methods for Structural RNAs

© L Ding, X Xue, S Lamarca, M Mohebbi, A Samad, RL Malmberg and L Cai

DOI: 10.15455/CMSR.2014.0003

Creative Commons BY-NC-ND 4.0 license
Table 4. Average performances of MC, Rosetta, NAST and BkTree, with results in two categories: average over all successfully resolved RNAs and average over all successfully resolved RNAs of length > 50. The best performance data are displayed in bold.

<table>
<thead>
<tr>
<th></th>
<th>All RNAs</th>
<th>All RNAs</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Success/Total</td>
<td>STY</td>
<td>PPV</td>
<td>MCC</td>
</tr>
<tr>
<td>MC</td>
<td>21/43</td>
<td>80.7</td>
<td>86.2</td>
<td>0.8344</td>
</tr>
<tr>
<td>Rosetta</td>
<td>43/43</td>
<td>62.8</td>
<td>80.3</td>
<td>0.7101</td>
</tr>
<tr>
<td>NAST</td>
<td>30/43</td>
<td>44.5</td>
<td>68.2</td>
<td>0.5508</td>
</tr>
<tr>
<td>BkTree</td>
<td>43/43</td>
<td>88.6</td>
<td>81.3</td>
<td>0.8482</td>
</tr>
</tbody>
</table>

Table 5. List of performance values predicted using MC, Rosetta and BkTree on 4 representative RNAs chosen by [15]. The results generated MC and Rosetta are obtained from the survey paper [14,15]. For every RNA, the best results are displayed in bold.

<table>
<thead>
<tr>
<th>PDB</th>
<th>Length</th>
<th>STY</th>
<th>PPV</th>
<th>MCC</th>
<th>RMSD</th>
<th>DI</th>
<th>STY</th>
<th>PPV</th>
<th>MCC</th>
<th>RMSD</th>
<th>DI</th>
<th>STY</th>
<th>PPV</th>
<th>MCC</th>
<th>RMSD</th>
<th>DI</th>
</tr>
</thead>
<tbody>
<tr>
<td>1KXK</td>
<td>70</td>
<td>81</td>
<td>89</td>
<td>0.849</td>
<td>9.49</td>
<td>11.16</td>
<td>74</td>
<td>85</td>
<td>0.793</td>
<td>17.22</td>
<td>21.69</td>
<td>97</td>
<td>94</td>
<td>0.9589</td>
<td>8.33</td>
<td>8.68</td>
</tr>
<tr>
<td>1XJR</td>
<td>42</td>
<td>76</td>
<td>87</td>
<td>0.8131</td>
<td>8.74</td>
<td>10.34</td>
<td>71</td>
<td>82</td>
<td>0.7676</td>
<td>11.63</td>
<td>15.21</td>
<td>91</td>
<td>84</td>
<td>0.8782</td>
<td>6.00</td>
<td>6.83</td>
</tr>
<tr>
<td>2QUS</td>
<td>71</td>
<td>76</td>
<td>92</td>
<td>0.8361</td>
<td>16.85</td>
<td>20.14</td>
<td>63</td>
<td>87</td>
<td>0.7403</td>
<td>18.10</td>
<td>24.72</td>
<td>92</td>
<td>86</td>
<td>0.8025</td>
<td>13.21</td>
<td>14.8</td>
</tr>
<tr>
<td>2QUS</td>
<td>69</td>
<td>78</td>
<td>86</td>
<td>0.819</td>
<td>18.41</td>
<td>22.44</td>
<td>58</td>
<td>86</td>
<td>0.7062</td>
<td>15.73</td>
<td>22.80</td>
<td>80</td>
<td>80</td>
<td>0.8</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>
Observations on the Feasibility of Exact Pareto Optimization with Applications to RNA folding

Cédric Saule and Robert Giegerich

Faculty of Technology and the Center for Biotechnology at Bielefeld University, Germany

cedric.saule@uni-bielefeld.de
r robert@techfak.uni-bielefeld.de

Abstract. Pareto optimization combines independent objectives by computing the Pareto front of its search space, defined as the set of all candidates for which no other candidate scores better under both objectives. This gives, in a precise sense, better information than an artificial amalgamation of different scores into a single objective, but is more costly to compute.

We define a general Pareto product operator $\ast_{\text{Par}}$ on scoring schemes. Independent of a particular algorithm, we prove that for two scoring schemes $A$ and $B$ used in dynamic programming, the scoring scheme $A \ast_{\text{Par}} B$ correctly performs Pareto optimization over the same search space. We show that a "Pareto-eager" implementation of dynamic programming can achieve the same asymptotics as a single-objective optimization which computes the same number of results. For RNA structure prediction under the minimum free energy versus the maximum expected accuracy model, we show that the empirical size of the Pareto front remains within reasonable bounds. Without artificial amalgamation of objectives, and with no heuristics involved, Pareto optimization is faster than computing the same number of answers separately for each objective.

Keywords: Pareto optimization, dynamic programming, RNA structure

1 Introduction

In combinatorial optimization, we evaluate a search space $X$ of solution candidates by means of an objective function $\psi$. Generated from some input data of size $n$, the search space $X$ is typically discrete and has size $O(\alpha^n)$ for some $\alpha$. Conceptually, as well as in practice, it is convenient to formulate the objective function as the composition of a choice function $\varphi$ and a scoring function $\sigma$, $\psi = \varphi \circ \sigma$, computing $(\varphi \circ \sigma)(X) = \varphi(\{\sigma(x) | x \in X\})$ as the overall solution. The most common form of the objective function $\psi$ is that $\sigma$ evaluates the candidates to a score (or cost) value, and $\varphi$ chooses the candidate which maximizes (or minimizes) this value. One or all optimal solutions can be returned, and with little difficulty, we can also define $\varphi$ to compute all candidates within a threshold of optimality. This scenario is the prototypical case we will base our discussion on. However, it should not go unmentioned that there are other, useful types of "choice" functions besides maximization or minimization, such as computing score sums, full enumeration of the search space, or stochastic sampling from it.
Multi-objective optimization arises when we have several criteria to evaluate our search space. Scanning the pizza space of our home town, we may be looking for the largest pizza, the cheapest, or the vegetarian pizza with the richest set of toppings. When we use these criteria in combination, the question arises exactly how we combine them.

Let us consider two objective functions $\psi_1 = \phi_1 \circ \sigma_1$ and $\psi_2 = \phi_2 \circ \sigma_2$ on the search space $X$ and let us define different variants of an operator $\ast$ to designate particular techniques of combining the two objective functions.

- **Additive combination** ($\psi_1 \ast_{+} \psi_2$) optimizes over the sum of the candidates scores employed by $\phi_1$ and $\phi_2$. This is a natural thing to do when the two scores are of the same type, and optimization goes in the same direction, i.e. $\phi_1 = \phi_2 =: \phi$; we define
  \[
  \psi_1 \ast_{+} \psi_2 = \phi \circ (\sigma_1 + \sigma_2). 
  \]
  (1)
  In fact, this recasts the problem in the form of a single objective problem with a combined scoring function. This applies e.g. for real costs (money), that sum up in the end no matter where they come from. Gotoh’s algorithm for sequence alignment under an affine gap model can be seen as an instance of this combination [6]. It minimizes the score sum of gap openings and gap extensions. However, often it is not clear how scores should be combined, and researchers resort to more general combinations.

- **Parametrized additive combination** ($\psi_1 \ast_{+\lambda} \psi_2$) is defined as
  \[
  \psi_1 \ast_{+\lambda} \psi_2 = \phi \circ (\lambda \sigma_1 + (1 - \lambda) \sigma_2). 
  \]
  (2)
  Here the extra parameter signals that there is something artificial in the additive combination of scores, and the $\lambda$ is to be trained from data in different application scenarios, or left as a choice to the user of the approach. Such functions are often used in bioinformatics [12, 5, 8, 11, 19]. For example, the Sankoff algorithm scores joint RNA sequence and folding by a combination of base pairing ($\psi_1$) and sequence alignment ($\psi_2$) score [12]. RNAalifold scores consensus structures by a combination of free energy ($\sigma_1$) and covariance ($\sigma_2$) scores [9]. Covariance scores are converted into “pseudo-energies”, and the parameter $\lambda$ controls the relative influence of the two score components.
  This combination often works well in practice, but a pragmatic smell remains. Returning to our earlier pizza space example: It does not really make sense to add the number of toppings to the size of the pizza, or subtract it from the price, no matter how we choose $\lambda$. In a way, the factor $\lambda$ manifests our discomfort with this situation.

- **Lexicographic combination** ($\psi_1 \ast_{lex} \psi_2$) performs optimization on pairs of scores of potentially different type, not to be combined into a single score.
  \[
  (\psi_1 \ast_{lex} \psi_2)(X) = (\phi_1, \phi_2)(\{(\sigma_1(x), \sigma_2(x))| x \in X\}), 
  \]
  (3)
  where $(\phi_1, \phi_2)$ optimizes lexicographically on the score pairs $(\sigma_1(x), \sigma_2(x))$. With the lexicographic combination, we define a primary and a secondary objective,
Exact Pareto Optimization with Applications to RNA folding

seeking either the largest among the cheapest pizzas, or the cheapest among the largest – certainly with different outcomes. This is very useful, for example, when \( \phi_1 \) produces a large number of co-optimal solutions. Having a secondary criterion choose from the co-optimals is preferable to returning an arbitrary optimal solution under the first objective, maybe even unaware that there were alternatives.

- **Pareto combination** \( (\psi_1 *_{Par} \psi_2) \) must be used in the case when there is no meaningful way to combine or prioritize the two objectives. It may also be useful and more informative in the previous scenario, producing a set of “optima” and letting the user decide the balance between the two objectives *a posteriori*. The solution set it computes is the **Pareto front** of \( X \). Taking \( \phi_1 \) and \( \phi_2 \) as maximization, the Pareto front operator \( pf \) is defined as

\[
\text{pf}(X) = \{(a, b) \in S \mid \not\exists (a', b') \in X \setminus \{a, b\} \text{ with } a \leq a', b \leq b'\}. \tag{4}
\]

In words: An element is in the Pareto front, if no other element scores strictly better in both dimensions. We define

\[
(\psi_1 *_{Par} \psi_2)(X) = \text{pf}\{\left(\sigma_1(x), \sigma_2(x)\right) \mid x \in X\}. \tag{5}
\]

The combinations \( *_+ \) and \( *_{+\lambda} \) are common practice and merit no theoretical investigation, as they reduce the problem to the single objective case. The lexicographic combination \( *_{lex} \) has been studied in detail in [17].

Let us now turn to non-heuristic cases of Pareto optimization. We are aware of only a few cases where it has been advocated within a dynamic programming approach. It was used by Getachew et al. [2] to find the shortest path in a network given different time cost/functions, computing the Pareto front. The Pareto-Optimal Allocation problem was solved with dynamic programming by Sitarz [16]. In the field of bioinformatics, Schnattinger et al. [15, 14] advocated Pareto optimization for the Sankoff problem. Their algorithm computes \( (\psi_1 *_{Par} \psi_2) \), where \( \psi_1 \) optimizes a sequence similarity score and \( \psi_2 \) optimizes base pair probabilities in the joint folding of two RNA sequences.

Pareto optimization in a dynamic programming approach brings about three specific problems:

(i) Does the Pareto combination of two objectives satisfy Bellman’s principle of optimality, the prerequisite for all dynamic programming?

(ii) How to compute Pareto fronts both efficiently and incrementally, when proceeding from smaller to larger sub-problems?

(iii) What is the empirical size of the Pareto front, compared to its expected size?

Heretofore, these issues had to be solved ad-hoc with every approach employing Pareto optimization. Motivated by and generalizing on the work by Schnattinger et al., we strive here for general insight in the use of Pareto optimization within dynamic programming algorithms.

The present article underlies the space constraints of a conference contribution. All proofs, algorithms, measurements, and further detail labeled “omitted” herein are available from the authors upon request. An extended manuscript including further experiments is in preparation.
Saule and Giegerich

2 Background

We introduce Pareto sets together with some basic mathematical properties and algorithms. We restrict our discussion to Pareto optimization over value pairs, rather than vectors of arbitrary dimension, which can be defined in an analogous way. This is justified by the fact that what we are aiming at is a general mechanism of combining two objective functions in a Pareto fashion, used with dynamic programming.

2.1 Basic properties of Pareto optimization

We start from two sets \( A \) and \( B \) and their Cartesian product \( C = A \times B \). \( A \) and \( B \) are totally ordered by relations \( >_A \) and \( >_B \), respectively. This induces a partial domination relation \( \succ \) on \( C \) as follows. \((a, b) \succ (a', b')\) if \( a >_A a' \) and \( b \geq_B b' \), or \( a \geq_A a' \) and \( b >_B b' \). In words, the dominating element must be larger in one dimension, and not smaller in the other. In \( X \subseteq C \), an element is dominant iff there is no other element in \( X \) that dominates it. We can now restate Eq. (4) in words as: The Pareto front of \( X \), denoted \( \text{pf}(X) \), is the set of all dominant elements in \( X \). The definition of \( \text{pf} \) actually depends on the underlying total orders, and we should write more precisely \( \text{pf}_{>_A,>_B} \), but for simplicity, we will suppress this detail until it becomes relevant.

It is easy to see that Pareto front is uniquely defined for all \( X \), that the operator \( \text{pf} \) is idempotent,

\[
\text{pf}(\text{pf}(X)) = \text{pf}(X)
\]

and that

\[
\text{pf}(X) \subseteq X
\]

holds by definition. Note that \( \text{pf} \) is not monotone with respect to \( \subseteq \).

A set \( X \subseteq C \) is a Pareto set if \( \text{pf}(X) = X \). Algorithmically, we represent sets as lists, without duplicate elements. If these lists have no particular order, we still call them sets.

A sorted list is a subset of \( C \) sorted lexicographically in decreasing order. If a sorted list holds a Pareto set, we call it a Pareto list. Naturally, on sorted lists we can perform certain operations more efficiently, which must be balanced against the effort of keeping lists sorted.

Observation 1 A Pareto list sorted on the first dimension based on \( >_A \) (i) is also sorted lexicographically in decreasing order, and at the same time (ii) is sorted based on \( >_B \) in increasing order.

This is true because when the list \( l \) is a Pareto list and \( (a, b) \in l \), there can be no other element \( (a, b') \) with \( b \neq b' \). Therefore, (i) the overall lexicographic order is determined solely by \( >_A \), and (ii) looking at the values in the second dimension alone, we find them in increasing order of \( >_B \).

This implies an observation on the size of Pareto fronts over discrete intervals:
Observation 2 If $A$ and $B$ are discrete intervals of size $N$, then any Pareto set over $A \times B$ has $O(N)$ elements.

This is true because each decrease in the first dimension must come with an increase in the second component. On random sets, the expected size of the Pareto front of a set of size $k$ follows the harmonic law, $H(k) = \sum_{i=1}^{k} (1/i)$ [7, 20].

By definition, the intersection of two Pareto sets is a Pareto set. This does not apply for Pareto set union, as elements in one Pareto set may be dominated by elements from the other. Therefore we define the Pareto merge operation

$$A \mathbin{\lor} B := \text{pf}(A \cup B)$$

(8)

Clearly, $\mathbin{\lor}$ inherits commutativity from $\cup$.

Observation 3: Pareto merge associativity.

$$(A \mathbin{\lor} B) \mathbin{\lor} C = A \mathbin{\lor} (B \mathbin{\lor} C)$$

(9)

Proof is omitted. As a consequence, we can simply write $A \mathbin{\lor} B \mathbin{\lor} C$.

2.2 Pareto operation complexity

We consider operations on Pareto sets and lists, where $k$ denotes their size. In dynamic programming, these operations are performed in the innermost loops of the algorithm, and hence they are crucial for overall efficiency. For algorithms implementing $\text{pf}$ see Section 5.1.

2.3 Pareto merge in linear time

We now specify an implementation of the Pareto merge operation $\mathbin{\lor}$ which makes use of the fact that its arguments are Pareto sets, represented as lists in decreasing order by the first component (and in increasing order by the second dimension).

$$[] \mathbin{\lor} y = y$$

$$x \mathbin{\lor} [] = x$$
Saule and Giegerich

\[(a, b) : x^p \quad (c, d) : y = \text{case } (a, b)?(c, d) \text{ of} \]
\[(> , >) \rightarrow (a, b) : (x^p \quad (\text{dropWhile}(\lambda(a, v).v \leq b), y)) \]
\[(> , =) \rightarrow (a, b) : (x^p y) \]
\[(> , <) \rightarrow (a, b) : (x^p ((c, d) : y)) \]
\[(= , >) \rightarrow (a, b) : (x^p \quad (\text{dropWhile}(\lambda(a, v).v \leq b), y)) \]
\[(= , =) \rightarrow (a, b) : (x^p y) \]
\[(=, <) \rightarrow (c, d) : ((\text{dropWhile}(\lambda(a, v).v \leq d), x)^p y) \]
\[(< , >) \rightarrow (c, d) : ((a, b) : x^p y) \]
\[(<, =) \rightarrow (c, d) : (x^p y) \]
\[(<, <) \rightarrow (c, d) : ((\text{dropWhile}(\lambda(a, v).v \leq d), x)^p y) \]

The function \text{dropWhile}(p, l) walks down a list \(l\) until it finds an element that does not satisfy the predicate \(p\). It returns this element and the remaining list in our case. We use it to eliminate elements smaller than \(b\) (resp. \(d\)) in the second dimension. At first glance, the combination of \(\vee\) and \text{dropWhile} reminds of an \(O(n^2)\) algorithm, but this is not true. For input lists of length \(n_1\) and \(n_2\), where \(n = n_1 + n_2\), the output list has at most length \(n\). It requires at most \(O(n)\) calls to \(\vee\). \text{dropWhile} requires \(k + 1\) calls when it deletes \(k\) elements, with \(k \in O(n)\). However, each element deleted by \text{dropWhile} safes a subsequent call to \(\vee\). Overall, the number of steps remains within \(O(n)\).

### 2.4 Pareto optimisation in dynamic programming

Dynamic Programming is governed by Bellman’s principle of optimality, which the objective function \(\psi = \varphi \circ \sigma\) must obey. Choice must distribute over scoring, which is computed incrementally from smaller to larger sub-solutions. For each incremental scoring function \(f\) that is used in the computation of \(\sigma\),

\[
\varphi(\{f(x), f(y)\}) = f(\varphi(\{x, y\}))
\]  

(10)

must hold. The above equation is formulated here for the simplest case: a unary function \(f\) and a choice function that returns a singleton result.

Dressing up Pareto optimization for dynamic programming, we must (i) formulate conditions under which it fulfills Bellman’s principle, and (ii) show how the Pareto front of the overall solution can be computed incrementally and efficiently from Pareto fronts of sub-solutions. Heretofore, these issues had to be resolved with every dynamic programming algorithm that uses Pareto optimization, such as the one by Sitarz or Schnattinger et al. [16, 15]. Striving for general results for a whole class of algorithms, we resort to the framework of algebraic dynamic programming.
3 Pareto optimization in ADP

Algebraic dynamic programming (ADP) is a framework for dynamic programming over sequential data [3]. Its declarative specifications achieve a perfect separation of the issues of search space construction, tabulation, and scoring, in sharp contrast to the traditional formulation of dynamic programming algorithms by matrix recurrences. Therefore, ADP lends itself to the investigation of Pareto optimization in dynamic programming in general, i.e. independent of a specific application domain.

Grammars and algebras describe ADP problems. See literature for theory and implementation [3, 13]. When grammar $G$ describes the search space, and algebra $A$ the scoring of candidates and the objective function $h$, an ADP algorithm is called in for input $x$ in the form $G(A, x)$, which is defined as

$$G(A, x) = h(\{A(t) \mid t \in L(G), \text{yield}(t) = x\}),$$

where $L(G)$ is the tree language generated by $G$, for a candidate $t$, $\text{yield}(t)$ is the string of leaf symbols from the underlying input alphabet, and $A(t)$ is its score under algebra $A$.

3.1 Relation between Pareto and other products

As we show next, Pareto optimization can rightfully be considered the most general of the combinations discussed here. This holds strictly in the sense that from the Pareto front, the solutions according to the other combinations can be extracted.

Theorem 1. Pareto front subsumption theorem For any grammar $G$, scoring algebras $A$ and $B$ satisfying Bellman’s principle, and input sequence $x$, consider the algebra combinations $A^+ B$, $A^\text{lex} B$, and $A^\text{Par} B$.

1. $G(A^+ B, x) = (\phi A^+ B)(G(A^\text{Par} B, x))$
2. $G(A^\text{lex} B, x) = (\phi A^\text{lex} B)(G(A^\text{Par} B, x))$

The proof is omitted.

3.2 Preservation of Bellman’s principle by the Pareto product

In this section we present our main theorem, showing that the Pareto product always preserves Bellman’s principle. For the Pareto product to apply, we have the prerequisite that algebras $A$ and $B$ both maximize over a total order. In this situation, Bellman’s principle specializes as follow:

Lemma 2. If $\phi$ maximizes over a total order, it implies that all $k$-ary functions $f$ for $k > 0$ are strictly monotone with respect to each argument.

Theorem 3. The Pareto product preserves Bellman’s principle.

Proof: Dominated solutions cannot become dominant by application of $f$. Details omitted.
4 Implementation

The Pareto product can be implemented simply by providing the Pareto front operator \( \varphi = \text{pf}_{\geq_B} \) as the choice function for the algebra product \( (A \ast \text{pf} B) \). This implementation can be improved by monitoring the status of intermediate results as lexicographically sorted lists.

We will describe these implementation issues by means of an example production which covers the relevant special cases. A tree grammar describing an ADP algorithm has an arbitrary number of productions, but their meaning is independent.

Let \( f, g, \) and \( h \) be a binary, a unary and a nullary scoring function from the underlying signature. A tree grammar rule such as

\[
W \rightarrow f < X, Y > | g < Z > | h <= 
\]

specifies the computation of partial results for a subproblem of type (i.e. derived from) \( W \) from partial results already computed from subproblems of types \( X \) and \( Y \), of type \( Z \), or for an empty subproblem via a (constant) scoring function \( h \). In general, signature functions may have arbitrary arity, and trees on the right-hand side can have arbitrary height. All this can be handled in analogy to what we do next.

We use the nonterminal symbols also as names for the list of subproblem solutions derived from them. Hence, we compute a list of answers

\[
W = [w_1, w_2, ...] 
\]

from

\[
X = [x_1, x_2, ...] 
\]

and so on. Note that \( h \) denotes a constant list, in most cases a singleton, but not necessarily so.

4.1 Standard implementation

Candidate lists are created by terminal grammar rules, by application of scoring functions to intermediate results, and by union of alternative answers from alternative rules for the same nonterminal.

We describe the standard implementation by three operators \( \otimes, \oplus, \# \), named “build”, “combine” and “select”, which are defined as follows:

\[
I \# \text{pf} = \text{pf}(I) 
\]

(11)

\[
I_1 \oplus I_2 = I_1 + I_2 
\]

(12)

\[
\otimes(f, X, Y) = [f(x, y) | x \in X, y \in Y] 
\]

(13)

\[
\otimes(g, X) = [g(x) | x \in X] 
\]

(14)

Hence, \( \# \) simply applies the choice function \( (11) \), generally \( \varphi \) and \( \text{pf} \) in our specific case. \( \oplus \) appends lists of solutions \( (12) \), and \( \otimes \) builds solutions for bigger subproblems from smaller ones \( (13, 14) \). Note that there is no requirement on the constant scoring function \( h \). Typically, such a function generates a single element anyway. In general however, it may produce a list of alternative answers, and this need not be a Pareto list in the standard implementation.

Using this set of definitions, our example production describes the computation of

\[
W = (\otimes(f, X, Y) \oplus \otimes(g, Z)) \oplus h \# \text{pf} 
\]

In section 5, we experiment with several implementations of \( \text{pf} \), for the standard case.
Exact Pareto Optimization with Applications to RNA folding

4.2 Pareto-eager implementation

The standard implementation applies a Pareto front operator after constructing a list of intermediate results. This list is built and combined from several sublists. By our main theorem, the Pareto front operation distributes over combinations of sublists, so we can integrate the # operator into the ⊕ operator. This has the potential that sizes of intermediate results are reduced as early as possible.

We define our operators as follows:

\[ l \# pf = l \]
\[ l_1 \oplus l_2 = l_1 \lor l_2 \]
\[ \otimes (f, X, Y) = \text{foldr}^\lor \left[ [f(x, y) | x \in X, y \in Y] \right] \]
\[ \otimes (g, X) = [g(x) | x \in X] \]
\[ h = pf(h) \]

Now the # operator effectively skips the computation of the Pareto front, as this is already performed at all other places. The \( \lor \) operation in Eq. (16) can assume argument lists to be Pareto lists already. In Eq. (18), the new list must be a Pareto list due to the strict monotonicity of \( g \). In Eq. (17), the same holds for each intermediate list \( [f(x, y) | x \in X] \) for each \( y \), and we can merge them successively. Here, \( h \) must produce Pareto lists as initial answers (Eq. (19)).

In Section 2.3 we showed that \( \lor \) can be implemented in \( O(N) \), and therefore, each step in the Pareto-eager implementation takes linear time. This means that Pareto-optimization requires no intrinsic overhead, compared to a single optimization scheme which returns a comparable number of results. This is an encouraging insight, but leaves us an aspect to worry about: The size \( N \) of the Pareto front which is computed from an input sequence of length \( n \).

4.3 Pareto front size

For a typical dynamic programming problem in sequence analysis, an input sequence of length \( n \) creates an exponential search space of \( O(2^n) \). Still, by tabulation and reuse of intermediate subproblem solutions, dynamic programming manages to solve such a problem in polynomial time, say \( O(n^r) \). The value of \( r \) depends on the nature of the problem, and when encoded in ADP, it is apparent as a property of the grammar which describes the problem decomposition [3]. We have \( r = 2 \) for simple sequence alignment, \( r = 3 \) for simple RNA structure prediction, \( r = 4 \) to \( r = 6 \) for RNA structures including various classes of pseudoknots, and so on. This all applies when a single, optimal result is returned.

For ADP algorithms returning the \( k \) best results, complexity must be stated more precisely as \( O(n^r k^{r-1}) \). As long as \( k \) is a constant, such as in \( k \)-best optimization, this does not change the asymptotics. However, computing all answers within \( p \) percent of
the optimal score may well incur exponential growth of $k$. With Pareto optimization, the size $k$ of the answer is not fixed in beforehand. The size of the Pareto front, for a set of size $N$, is expected to be $H(N)$ (cf. Section 2.1). Using $N \in O(2^n)$ and $H(N) \approx \log(N)$ [7], we can expect an effective size of the result sets in $O(n)$. Taking all things together, we can compute the Pareto front for an (algebraic) dynamic programming problem in $O(n2^{-r})$ expected time, where $n$ is input length and $r$ reflects the complexity of the search space.

In applications, the size of the Pareto front need not follow expectation. We may achieve efficiency of $O(n'k'^{-1})$ where $k' \ll n$. Fortunately, in the application scenario of the next section, we find ourselves in this positive situation.

5 Applications

In this section, we report observations from different evaluations. (1) We use two real-world applications as black boxes, to measure runtime and space performance of Pareto optimization. (2) We use the same programs for assessing the empirical size of the Pareto front. Our test data consists of 331 RNA sequences of length 12 to 356 nucleotides, extracted from the full data set used in [1].

5.1 Algorithms implemented

We choose as test cases RNA folding algorithms, namely minimum free energy folding (MFE) and maximum expected accuracy folding (MEA). For each application, we can re-use grammars and algebras from the RNAshapes repository [10].

For the crucial operation $\text{pf}$, we tested several implementations. Let $k = |X|$. The insertion phase adds an element to the end of the result list, if it is not dominated by any other element already in that list. This asymptotically in $O(k^2)$, and the worst case $\text{pf}_{\text{naive}}$ actually occurs when $X$ is already a sorted Pareto list!

- $\text{pf}_{\text{naive}}(X)$ computes a sorted result list. It requires $O(k)$ steps if $X$ is sorted, and $O(k^2)$ steps in general.
- $\text{pf}_{\text{sort}}$ also computes a sorted result list. It first sorts $X$ lexicographically in $O(k \log k)$ steps, and from the result, obtains the Pareto front in $O(n)$, giving $O(k \log k)$ overall.

5.2 Runtime and memory requirements.

We evaluate the performance of the Pareto front computation, using $\text{pf}_{\text{naive}}(X)$, $\text{pf}_{\text{smooth}}(X)$, $\text{pf}_{\text{sort}}(X)$, and $\text{pf}_{\text{isort}}(X)$. Note that all compute the same Pareto front, and hence have the same $k$ in their asymptotics. For a fair comparison with two single-objective algorithms MFE and MEA, we use their versions $\text{MFE}(k)$ and $\text{MEA}(k)$, computing the $k$ best structures. Here, $k$ is set to the actual Pareto front size for the given input (which, of course, is only known because before we compute the Pareto front with the other
Exact Pareto Optimization with Applications to RNA folding algorithms. All programs are compiled by the Bellman’s GAP compiler [13] using the same optimization options.

In Table 1 we show computation time and memory consumption, accumulated over all sequences and specifically for the longest sequence. These are our main observations:

<table>
<thead>
<tr>
<th>Algebra</th>
<th>Time (min)</th>
<th>Memory (GB)</th>
<th>Time (min)</th>
<th>Memory (GB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>MFE(k) alone</td>
<td>71</td>
<td>163.68</td>
<td>5</td>
<td>1.16</td>
</tr>
<tr>
<td>MEA(k) alone</td>
<td>61</td>
<td>153.51</td>
<td>5</td>
<td>1.05</td>
</tr>
<tr>
<td>MFE(k) + MEA(k)</td>
<td>132(+      163.68(max)</td>
<td>10</td>
<td>1.16 (max)</td>
<td></td>
</tr>
<tr>
<td>(MFE * Par MEA)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>pf\text{naive}</td>
<td>8</td>
<td>197.28</td>
<td>0.22</td>
<td>1.28</td>
</tr>
<tr>
<td>pf\text{smooth}</td>
<td>9.5</td>
<td>192.79</td>
<td>0.5</td>
<td>1.28</td>
</tr>
<tr>
<td>pf\text{sort}</td>
<td>18</td>
<td>271.21</td>
<td>1</td>
<td>1.28</td>
</tr>
<tr>
<td>pf\text{isort}</td>
<td>32</td>
<td>250.21</td>
<td>3</td>
<td>2.11</td>
</tr>
</tbody>
</table>

Table 1. Runtimes and memory requirements for MFE(k), MEA(k) (where k is the empirical Pareto front size for a given input), and their Pareto product (MFE \* Par MEA), accumulated over 331 sequences (left) and for the longest sequence (n = 356, k = 38, right). The computations were performed by using Bellman’s Gap.

(1) In terms of runtime, we find that the Pareto optimization performs not only better than the sum of the two independent optimizations, but also better than each of them individually. We attribute this to the fact that the Pareto algorithm adjusts itself to the size of the Pareto front, and this size tends\(^1\) to be smaller than k for small subproblems. The search space itself, however, is exponentially larger than the Pareto front, and even on small sub-words it provides k near-optimals for MFE(k) and MEA(k) to spend computation on. This effect is strongest for our longest sequence, where k = 38 and the ratio of (MFE(k) + MEA(k))/pf\text{naive} \approx 45.

(2) The average case behaviour of pf\text{naive}(X) is superior to all the sorting implementations of pf. This is an unexpected and interesting observation. We attribute this to a positive randomization effect. Comparing a new element to the extremal points of the Pareto front, maximal in one but minimal in the other dimension, is unlikely to establish domination. This what always happens first with sorted intermediate lists, and the element will walk along towards the middle of the list until it eventually is found to be dominated. In unsorted lists, a non-extremal element that dominates the new entry will, on average, be encountered earlier.

(3) Memory consumption of Pareto optimization is consistent over different implementations of pf. It is higher than either MFE(k) or MEA(k) alone, but clearly less than the sum of MFE(k) and MEA(k). This is better than expected, because after all, it solves both problems simultaneously.

\(^1\) This is only a tendency – a final Pareto front of size k does not preclude intermediate results with Pareto fronts larger than k.
Saule and Giegerich

Fig. 1. a) Empirical Pareto front size of $OverDangle(MFE \ast_{Par} MEA, x)$ as a function of $|x|$. The red line corresponds the $H(|X|)$, the expected Pareto front size according to the harmonic law, applied to the empirical value of $|X|$ for each $x$. b) Number of abstract RNA shapes [4] in the Pareto front, in function of the sequence size.

5.3 Pareto front size

The size of the Pareto front is of critical practical importance. Pareto front sizes in the hundreds, even for sequences of moderate length, would be prohibitive. This question can hardly be assessed analytically. Figure 1 shows our measurements. We observe the following:

- Much to our relief, Pareto front sizes are quite moderate, ranging round 10 for $n = 100$, 15 for $n = 200$, up to 45 for $n = 274$. Specifically, our longest sequence ($n = 356$) has a Pareto front of size 38.
- Variance is high (as expected), and because of the strong variation, we did not fit a line through our measurement points. However, they are all dominated by the expected size of the Pareto front (red line).
- We did not smooth the graph for $H(|X|)$, such that it also demonstrates the variance in the search space sizes; just read the y-axis as a logarithmic scale for $\mu^H(|X|)$. The roughly linear behavior conforms with the theoretical analysis.

Abstract shape analysis is another established method (unrelated to Pareto optimization) to obtain an interesting set of near-optimal solutions from the search space [18]. We checked the ratio of the number of structures in the Pareto front, and the number of different abstract shapes they represent, but this ratio, ranging from 1 to $\approx 12$, did not exhibit an obvious pattern. An experiment with $OverDangle(shape \ast (MFE \ast_{Par} MEA), x)$ is yet to be performed.

© C Saule and R Giegerich

DOI: 10.15455/CMSR.2014.0004

Creative Commons

BY-NC-ND 4.0 License
Exact Pareto Optimization with Applications to RNA folding

6 Conclusion

We have shown that the exact Pareto front of two independent objectives can be computed by dynamic programming. The theoretical prerequisite for this is the preservation of Bellman’s principle by the Pareto combination operator $\ast_{\text{Par}}$, established in our main theorem.

We have shown that by the Pareto-eager implementation, one can achieve Pareto optimization without an intrinsic penalty, compared to other optimizations which return a comparable number of results.

We have shown that empirically, for the case of RNA folding under different objectives, the size of the Pareto front remains within moderate bonds, clearly lower than theoretical expectation.

All in all, this says the Pareto optimization is practical for sequence analysis and moderate sequence sizes.

Acknowledgements

The authors would like to thank T. Schnattinger and H.A. Kestler for the discussions which inspired this generalization of their work. Thanks go to Stefan Janssen for help with the Bellman’s GAP system, and to the anonymous CMSR reviewers for helpful feedback.

References

Saule and Giegerich


Recent Results on Three Problems in Comparative Structural RNAomics

Shay Zakov, Nimrod Milo, Tamar Pinhas, Sivan Yogev, Erez Katzenelson, Eitan Bachmat, Yefim Dinitz, Dekel Tsur, and Michal Ziv-Ukelson

Department of Computer Science, Ben-Gurion University of the Negev.
Email: {zakovs,milon,tamar,yogev,erez,ebachmat,dinitz,dekelts,michaluz}@cs.bgu.ac.il

Abstract. We review our recent results on three problems in Comparative Structural RNAomics. Our contribution includes: (1) a new worst-case bound for Discrete RNA Folding, (2) Unordered Unrooted Comparisons of RNA Trees, and (3) an RNA Homology Search where the query is an RNA sequence and the output consists of Sequence-Structure homology hits allowing pseudoknots and alternative stems.

A common denominator of these three works is that they demonstrate research problems tackled within current structural RNAomics, whose solution extends to more general classical problems in Computer Science, yielding respectively: (1) a new theoretical bound for Discrete Min-Plus Matrix Multiplication, (2) a new theoretical bound for All-Pairs Cavity Bipartite Matching, and (3) new admissible heuristics to speed up Max Weighted Clique. Source code and web-interface for the tools can be found in our website http://www.cs.bgu.ac.il/~negevcb/.

A New Theoretical Time Bound for Discrete RNA Folding. The Classical 2D RNA Folding problem (without pseudoknots and assuming a constant bound on internal loop size) is a special form of the weighted Context Free Grammar (CFG) Parsing optimization problem. There are previous works that speed up CFG decision algorithms: Valiant’s approach via Fast Boolean Matrix Multiplication, and Graham’s via a boolean variant of the Four-Russians approach. There are RNA motivated extensions of these works to the weighted optimization problem: Akutsu extended Valiant’s approach to yield an $O(n^3 \log^2 n \log \log n)$ time result for RNA Folding, and Frid and Gusfield obtained an $O(n^3 / \log n)$ result for Discrete RNA Folding, extending Graham’s Boolean method to fast max-plus multiplication of vectors in which differences between adjacent values are confined to a small integer interval.

Recently, while studying another weighted CFG problem variant (Edit Distance with Duplications and Contractions) [3], we obtained a new bound for the discrete case of this problem. Our algorithm is an adaptation of Williams’ algorithm for finite semiring matrix-vector multiplication, combined with some notions similar to the approach employed by Frid and Gusfield’s algorithm. It follows the concepts of the Four-Russians approach of tabulating recurring computations. The new $O(n^3 / \log^2 n)$ time algorithm

* This research was partially supported by ISF grant 478/10 and by the Frankel Center for Computer Science at Ben Gurion University of the Negev.
Zakov, Milo, Pinhas, Yogev, Katzenelson, Bachmat, Dinitz, Tsur and Ziv-Ukelson

applies to several problems from the domains of discrete context-free grammar parsing and RNA folding and, in particular, implies the currently asymptotically fastest algorithm for single-strand RNA folding with discrete cost functions.

**Unrooted Unordered Comparison of RNA Trees.** A mainstream approach to (pseudoknot free) RNA secondary structure comparison represents the structures as trees, and applies tree alignment algorithms to their comparison. Currently available bioinformatic softwares for RNA tree comparison usually apply rooted ordered tree alignment. However, there are known evolutionary events, such as segment insertions, translocations and reversals, which could be modeled as a reordering or re-rooting of branches in the corresponding trees. This motivates algorithms for Unordered Unrooted Alignment of RNA Trees.

Due to NP hardness of general Unordered Tree Edit Distance, we define a Homeomorphic Subtree Alignment variant, in which only nodes of degree 2 (or whole subtrees) can be deleted from both trees [2]. We propose and implement an efficient algorithm for this new problem variant. For the most general unrooted unordered case, the time complexity of our algorithm is $O(n^3)$, where $n$ denotes the number of nodes in the compared trees. This improves the time complexity of previous algorithms for less general variants of the problem.

**A New RNA Sequence/Structure Homology Search.** A practical problem in structural RNAomics is that of genome-scale Sequence-Structure Search for conserved homologues of a given RNA sequence, when the structural conservation criteria are general enough to consider pseudoknots and potential dynamic alternative stem configurations.

We propose a new search engine based on a structural representation of an RNA sequence by its potential stems [1]. Potential stems in genomic sequences are identified in a preprocessing stage, and indexed. A user-provided query sequence is likewise processed, and stems from the target genomes that are similar to the query stems are retrieved from the index. Then, relevant genomic regions are identified and ranked according to the highest scoring mapping to be found between a subset of their stems versus a subset of the query stems, where the (one-to-one) stem mapping across the sets enforces conservation of cross-stem topological relations (Nested, Crossing, Adjacent, or Overlapping) within the sets. This search yields a new, NP hard, weighted variant of 2-interval pattern matching, which we solve via an efficient reduction to Max-Weighted-Clique.

**References**

Scalable structural clustering of local RNA secondary structures

Steffen Heyne, Fabrizio Costa, Dominic Rose and Rolf Backofen

Bioinformatics Group, Department of Computer Science, University of Freiburg, Georges-Köhler-Allee 106, D-79110 Freiburg, Germany.

1 Introduction

It is increasingly evident that the eukaryotic genome is pervasively transcribed [4] and that non-protein coding RNAs (ncRNAs) play a key role in regulating gene expression and cell biology [3], affecting for instance (post-)transcriptional regulation, chromatin-remodelling, differentiation and development. In all kingdoms of life, genome-wide analysis are currently identifying a large numbers of potential ncRNAs (up to 450.000 only in the human genome [11]) and therefore the prediction, comparison, and functional annotation of ncRNAs are major tasks of current RNA research. Such studies identify genomic loci that are under stabilizing selection and that exhibit thermodynamically stable secondary structures and that therefore constitute prime candidates for novel functional ncRNAs. The functional annotation task is however a complex endeavor since, in contrast to protein-coding genes, ncRNAs belong to a large number of diverse classes with vastly different structures, functions, and evolutionary patterns [2].

The current classification system divides ncRNAs into (a) families according to functional, structural, or compositional similarities (the Rfam database lists as of today more than 2000 RNA families [6]), and (b) into RNA classes that group ncRNAs whose members have no discernible homology at the sequence level, but that have common structural and functional properties (e.g. snoRNAs and micro RNAs). For these reasons clustering according to sequence-structure similarity is nowadays the de-facto standard for ncRNA annotation. The quality and computational complexity of the clustering procedure depend on the underlying pairwise sequence comparison method. The most generic methods (LocARNA [17] and FOLDALIGN [16]) use derivatives of the full Sankoff algorithm [13] of simultaneous alignment and folding, but can be used only on small sets given their complexity (at least $O(n^4)$).

In order to achieve a reasonable trade-off between time and quality, many approaches use different heuristics: (a) using simplifications in the structural model, or (b) using sequence information as prior knowledge to speed up the computation. In the first case ([12] and [9]) one predicts structures for each individual sequence (a task that is known to be error prone). In the second case sequences are first clustered by sequence-alignment [15,10] and then conserved consensus structures are predicted (using RNAALIFOLD [1] or PETFOLD [14] for example). The major problem here is that ncRNA sequences evolve much faster than their structure, to the extent that often no homology on the sequence level is detectable (family assignments of sequence alignments at pairwise sequence identities below 60% are often wrong [7]).
2 Contribution

Here, we propose an alignment-free approach for clustering RNA sequences according to sequence and structure information. We extend a fast graph kernel technique that we have developed [5] for chemoinformatics applications and we adapt it to detect similarities between RNA secondary structures. The key novelties are twofold: (1) we represent multiple folding hypothesis associated to a single RNA sequence in a flexible graph format; and (2) we efficiently convert the graph encoding into a very high dimensional sparse vectors. The first strategy allows us to compensate the inaccuracies of the minimum free energy solution. The second strategy allows us to use locality sensitive hashing methods to identify clusters with a complexity that is linear in the number of sequences $N$, i.e. avoiding the quadratic complexity arising from pairwise similarity computations.

We have integrated the approach in a ready-to-use pipeline for large-scale clustering of putative ncRNA. The method has been evaluated on known ncRNA classes and compared against existing approaches such as LocARNAand RNASOU[8]. We show that not only we obtain clusters of high quality, but also we achieve striking speedups: from years to days for serial computation, down to hours when considering the parallel implementation.

We applied our method to six heterogeneous large-scale data sets containing more than 220,000 sequence fragments in total. We have analyzed predicted short ncRNAs which were lacking reliable class assignments and we have searched for local structural elements specific to experimentally validated lincRNAs. In this latter case we found enriched GO-terms for lincRNAs containing predicted local motifs that suggest a connection to vital processes of the human nervous system.

References

6. Paul P. Gardner, Jennifer Daub, John Tate, Benjamin L. Moore, Isabelle H. Osuch, Sam Griffiths-Jones, Robert D. Finn, Eric P. Nawrocki, Diana L. Kolbe, Sean R. Eddy, and Alex
Scalable structural clustering of local RNA secondary structures


The role of periodic mRNA secondary structure and RNA-RNA interactions in biological regulation and complexity

Svetlana A. Shabalina 1, Aleksey Y. Ogurtsov 1, Anna Kashina 2, and Nikolay A. Spiridonov 3

1 National Center for Biotechnology Information, National Library of Medicine, National Institutes of Health, Bethesda, MD 20894
2 Department of Animal Biology, School of Veterinary Medicine, University of Pennsylvania, Philadelphia, PA 19104, USA
3 Division of Therapeutic Proteins, Center for Drug Evaluation and Research, US Food and Drug Administration, Bethesda, MD 20892

Shabalin@ncbi.nlm.nih.gov

mRNA carries a wealth of the structural and regulatory information in addition to the encoded amino acid sequence. This information defines mRNAs secondary structure and stability, pre-mRNA splicing efficiency, regulates rate of translation and affects folding and posttranslational modifications of the nascent polypeptide (1-3). Emerging evidence suggests important biological functions for synonymous nucleotides and “silent” mutations in the protein coding genes that do not change the amino acid sequences of the proteins.

Using our software Afold, we performed the first transcriptome-wide analysis of the mRNA folding in different organisms, and demonstrated that the structure of the genetic code and the unequal use of synonymous codons create a periodic pattern of nucleotide involvement in mRNA secondary structure in the protein coding regions (CDSs) (4-5). We also showed how RNA secondary structure might regulate gene expression and suggested that a periodic pattern in the CDS is likely responsible for translation frame monitoring (Figure 1 - low centre panel, 1-2). The degenerate codon sites make the greatest contribution to mRNA stability. Our results support the hypothesis that redundancies in the genetic code enable mRNA sequences to satisfy requirements for both protein and RNA structure, and suggest that selection in favor of G and C may be operating in synonymous codons to maintain a more stable and ordered mRNA secondary structure, which is likely to be important for transcript stability and translation. Functional domains of the mRNA (5’UTR, CDS and 3’UTR) preferentially fold onto themselves, while domain boundaries are characterized by relaxed secondary structures, as compared to the overall mRNA folding. Relaxed secondary structures in the vicinity of the start and stop codon regions could facilitate the initiation and termination of translation. Comparative analysis of mRNA secondary structure patterns for eukaryotes and prokaryotes revealed the ubiquity of periodic mRNA secondary structures and RNA level selection pressure acting at the level of mRNA secondary structure in different organisms.

Systematic differences in selection pressure exist between synonymous and non-synonymous positions in mRNA coding regions (2). Selection pressure on the coding gene regions follows a three-nucleotide periodic pattern of nucleotide base pairing in...
Fig. 1. Periodic pattern of nucleotide involvement in secondary structure formation and sequence conservation around the start and the stop codons in human mRNAs. Positions from -30 to -1 correspond to 5'-UTRs and positions from 1 to 60 correspond to CDSs (upper left panel). Positions from -60 to -1 correspond to CDSs and positions from 1 to 30 correspond to 3'-UTRs (upper right panel). Blue, sequence conservation in 6919 orthologous human and mouse mRNAs. Red, base-paired nucleotides in 19 317 human mRNAs. Green, free Gibbs energy of base-pairing in 19 317 human mRNAs. Structural features of the untranslated regions (UTRs) and coding sequences (CDSs) have a major role in the control of mRNA translation. The relaxed secondary structures in UTRs are common for many mRNAs and involved in regulation of initiation (low left panel) and termination (low right panel) of translation. Periodic pattern in the CDS is likely responsible for translation frame monitoring (low center panel). Groups of genes with distinct levels of expression are presented in different colors. This figure is adapted from Shabalina et al., 2013.

mRNA that is imposed by the genetic code, where synonymous positions of the coding regions have higher hybridization potentials and are multifunctional in their regulatory role and structural functions (1, 4-5; Figure 1). Our theoretical estimations of the periodic patterns for eukaryotic genomes are in good agreement with recently published results of experimental genome-wide profiling of the RNA secondary structure in human, yeast and plants (6-8). The trio of experimental reports provides the first insight into the mRNA secondary structure of an entire transcriptome in eukaryotes in vivo and lends support to three-nucleotide structure periodicity in the CDS, and its absence in UTRs, which were demonstrated in our prior computational predictions. Our data to-
Periodic RNA structure and interactions in regulation and complexity

gather with recent experimental evidence suggests that there is an evolutionary tradeoff between selective pressure acting at the RNA and protein levels (2). The RNA level selection pressure is common for both prokaryotes and eukaryotes and more widely distributed in nature than previously thought (1, 4-5), suggesting that periodic structure may have evolved as a universal regulatory feature of translated portions of mRNAs (1, 4-6) which can support maintenance of the reading frame in protein coding regions and influence gene expression (1, 4).

Degenerate codon sites are important for maintaining a more ordered and stable mRNA secondary structure in the protein coding regions and supporting maintenance of the reading frame in protein coding regions. It is likely that direct base-pairing of particular mRNAs to rRNAs (“clinger” elements) within ribosomes may provide a mechanism of translational control where clinger elements may act both as up-regulating and down-regulating functional sites (1).

References

Searching for SNPs disrupting RNA secondary structures (Keynote)

Jan Gorodkin

Center for Non-Coding RNA in Technology and Health, University of Copenhagen, Danemark

Single Nucleotide Polymorphisms (SNPs) can have large impact on diseases as well as phenotypic traits. Traditionally, SNPs have been studied in protein coding sequence and lately also in regulatory elements such as transcription factor binding sites. Since phenotypic SNPs are widespread in the genome it is of equal interest to search for their impact everywhere including in RNA structure in transcriptomic sequence. Studying the potential impact of, for example, SNPs in coding sequence takes outset in non-synonymous changes and these have then further been used to study structure disruptions which then again are used to imply functional changes. In contrast, studying SNPs for structure disrupting potential in RNA is more complex, because longer range base pairings often are involved.

A number of strategies have been employed to address this, but they have mainly considered the RNA sequence globally, and thus local changes in large sequence can be harder to detect. We address this by constructing a computational approach, called RNAsnp, which considers the sequences locally from globally computed base pair probabilities in either the full sequence or in sliding windows. Our approach compares the wild-type and mutant sequences and search for the region which maximizes the difference in base pair probabilities using a given distance measure. Furthermore, we compute mutation effects by empirical p-values.

On the analysis of disease-associated SNPs in UnTranslated Regions (UTRs) we obtain substantially more candidates (20 vs. 3) than obtained by a global strategy on a set of 501 diseases associated SNPs. In a further study of cancer associated Single Nucleotide Variants (SNVs), we combined prediction of disrupted local RNA secondary structure and microRNA targets. We analyzed existing transcriptome data from patients with non-small cell lung cancer (NSCLC). In the original set, aimed at finding non-synomous SNVs, 40% of the in total (somatic and germ-line) 73,717 SNVs overlap UTRs. Of 29,290 SNVs in UTRs of 6,462 genes, we predict 962 (408 associated with local RNA structure; 490 to miRNA targets) disruptive SNVs in 803 different genes. Of these 188 (23.4%) were previously known to be cancer associated, which is significantly higher (p=0.032) than the ratio of 1,347 of 6,462 in the full data set. This analysis can furthermore be used for network analysis indicating where disruptive SNVs appear.

RNAsnp is available as standalone software and as webserver at:

http://rth.dk/resources/rnaspn
Deciphering the regulatory functions of miRNAs
(Keynote)

Mihaela Zavolan
Biozentrum, University of Basel, Switzerland

miRNAs are small RNAs that guide Argonaute proteins to target mRNAs through perfect complementarity involving 7-8 nucleotides of the miRNA's 5' end. These canonical targets typically undergo degradation and translational inhibition. However, recent studies have suggested that miRNA target degradation can give rise to additional behaviors. These include the threshold-linear response of the targets to their transcriptional induction, reduction of the noise in target expression and induction of correlations in the expression of the targets of a given miRNA. Here I will discuss our combined, experimental and computational, approaches towards predicting miRNA targets and characterizing the functional impact of miRNAs.
Index of Authors

Allali, Julien, 1
Bachmat, Eitan, 57
Backofen, Rolf, 59
Bourgeade, Laetitia, 1
Cai, Liming, 25
Chauve, Cédric, 1
Costa, Fabrizio, 59
Ding, Liang, 25
Dinitz, Yefim, 57
Giegerich, Robert, 43
Giraud, Mathieu, 13
Gorodkin, Jan, 65
Heyne, Steffen, 59
Kashina, Anna, 62
Katzenelson, Erez, 57
Lamarca, Sal, 25
Malmberg, Russell L., 25
Milo, Nimrod, 57
Mohebbi, Mohammad, 25
Ogurtsov, Aleksey, 62
Pinhas, Tamar, 57
Rose, Dominic, 59
Saffarian, Azadeh, 13
Samad, Abdul, 25
Saule, Cédric, 43
Shabalina, Svetlana, 62
Spiridonov, Nikolay, 62
Touzet, Hélène, 13
Tsur, Dekel, 57
Xue, Xingran, 25
YogeV, Sivan, 57
Zakov, Shay, 57
Zavolan, Mihaela, 66
Ziv-Ukelson, Michal, 57