Media Engineering and Technology Faculty
German University in Cairo

A new heuristic algorithm for IntaRNA for improved RNA-RNA interaction prediction

Bachelor Thesis

Author:	Mohamed Ezz El-Din El-Shaer
Supervisor:	Prof. Dr. Rolf Backofen
	Andreas Richter
Reviewer:	Prof. Dr. Slim Abdennadher
Submission Date:	4 March, 2011

This is to certify that:

(i) The thesis comprimises only my original work toward the Bachelor Degree

(ii) Due acknowlegement has been made in the text to all other material used

Mohamed Ezz El-Din El-Shaer
4 March, 2011

Acknowledgments

I would like to thank Prof. Dr. Rolf Backofen so much for offering me this great opportunity to work on my Bachelor Thesis in his group of Bioinformatics in Albert Ludwigs University of Freiburg.

I owe alot to my supervisor Andreas Richter, this thesis would not have been possible without his continuous guidance and support with both knowledge and ideas throughout the research. His analytical skills and professional attitude affected me personally. I want to thank him alot for all his efforts, working with him was a great pleasure for me.

I would like to show my deep gratitude to Prof. Dr. Slim Abdennadher for arranging this opportunity.

I am also grateful to my respectful parents for their great support along my stay, without whom I wouldn't have been able to stay abroad.

I want to thank everybody in the group for being helpful and friendly, which contributed positively to my work.

Abstract

The number of discovered ncRNAs(non-coding RNAs) that regulate target mRNAs by base pairing is growing fast. This demands for identification of the target mRNAs for those ncRNAs. Thus prediction of such interactions between ncRNAs and mRNAs became of great neccesity to help identify targets for known ncRNAs. A few computational algorithms for this purpose were developed to predict such interactions. While some of the algorithms were fast enough for genome-wide searches, they were not so accurate in predicting interactions between long RNAs. This is because they neglected an important factor for interaction formation which is the interacting site accessibility. *IntaRNA* considers site accessibility while maintaining the same time and space complexities of these fast algorithms. *IntaRNA* includes two algorithms, one that gives optimal results according to the *Turner free* energy model[1], but is time consuming with time complexity $O(n^2m^2)$. The second algorithm is heuristic with time complexity $O(nm)$ only, but does not give optimal results for all input sequences. In this thesis we present improvements over both algorithms of *IntaRNA*. First we modified the non-heuristic algorithm to model more accurately how RNAs are actually forming an interaction. It simulates - in the same order - the sequence of events in which interaction formation is thought to happen in real. The new implementation allows to forbid high energy barriers that might be encountered during interaction formation and that are less likely to be overcome. Second we improved the accuracy of the heuristic algorithm of *IntaRNA*, making it more accurate and reliable for use in biological researches, without significantly increasing its runtime and space requirements.

[1] The energy model used scores interactions by considering both hybridization and accessibility energies

Contents

Chapter 1

Introduction

In this chapter we introduce the problem of RNA-RNA interaction prediction and its importance to the field of Bioinformatics. Then we discuss some of the previous approaches for the problem.

1.1 General motivation

For a long time RNA was thought to be involved only in protein synthesis process. However, in the last two decades many non-protein coding RNAs (ncRNAs) were discovered. Some of them function as post-transcriptional regulators by binding to mRNAs (messenger RNAs) via base pairing, like for example miRNAs (micro RNAs), siRNAs (small interfering RNAs) or bacterial sRNAs (small RNAs). Although many ncRNAs have been discovered and verified, only few target RNAs for these ncRNAs are currently known. Knowing these targets helps assign a function to these ncRNAs. Since target identification is tedious, expensive and time consuming to be accomplished experimentally, prediction algorithms of such interactions became of great neccesity. Using computation power can help to identify targets for known ncRNAs in a more convenient and efficient way. Finding which RNAs can have a stable interaction with the known ncRNAs is actually finding the target mRNAs for those ncRNAs. *IntaRNA* is an algorithm for this purpose.

1.2 Problem statement

Two RNA sequences are given, presumably target RNA sequence and ncRNA sequence, represented as strings S^1 and S^2, respectively. Each character in the strings represents one nucleotide. An interaction can be defined as a set S of pairs (x, y), such that x is a position of a nucleotide in S^1 and y is a position of a nucleotide in S^2. Each pair in the set represents a base pairing between the base at position x in the target RNA and the base at position y in the ncRNA, with the condition that a position x or y cannot exist in more than one pair. An energy value is assigned to interacting regions according to the *Turner Free* energy model.

The problem is to find the best interaction between the two sequences energy-wise, that is, compute the interaction with the minimum free energy (MFE).
This problem is known as *RNA-RNA Interaction Prediction Problem RIP* and it was proven in [1] to be an NP-complete if we want to consider all possible interaction structures. This means that an exact algorithm for the problem would require exponential time. However polynomial time algorithms exist when we restrict the class of interactions to be considered. Different algorithms handle the problem with different approaches and most importantly they consider different subsets of all possible interactions to avoid the exponential time requirement. In the next section a brief overview of some algorithms is presented.

1.3 Previous work

1.3.1 *RNAcofold, PairFold*

Both the *RNAcofold* [6] and the *PairFold* [2] algorithms use nearly the same idea of concatenating the two input sequences S^1 and S^2 to one sequence (denoted as S^{12}), then the concatenated sequence S^{12} is folded like a single RNA sequence using any algorithm like the classical *Zuker* folding algorithm for example, while keeping track of the concatenation point.

Although the idea is nice and very simple, it has the major drawback that it does not consider the case when there are pseudoknots in S^{12}, which is very likely to happen in RNA-RNA interactions and should be considered because S^{12} is actually two separate sequences binding together. To clarify more, consider the example in Figure 1.1, when one of the sequences is binding to a hairpin loop in the other sequence.

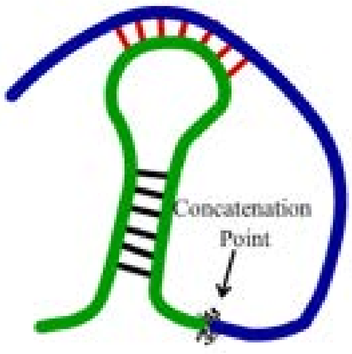


Figure 1.1: The structure shown appears to *RNAcofold* and *PairFold* as a pseudoknot in S^{12} so it is not considered as one of the possible solutions.

Speaking about these two sequences as one sequence, then this interaction is pseudoknotted , which the RNA folding algorithms used by *RNAcofold* and *PairFold* never consider. While speaking about them as two separate RNA sequences , which is the case, then the interaction has a common structure because interaction sites are often located in loop regions. Thus this kind of structures cannot be handled by these algorithms.

1.3.2 *RNAhybrid*

RNAhybrid [25] is an RNA-RNA interaction prediction algorithm that minimizes the hybridization energy, i.e., the energy of the base pairings between the two sequences. It neglects an important factor for having a stable binding between the two sequences, which is the accessibility of the interacting regions in the sequences. Accessibility energy is the amount of energy required to make the the interacting regions in each sequence single stranded, or in other words, free from intramolecular pairings. Although the algorithm is fast with time complexity $O(nm)$, neglecting the accessibility factor makes it unreliable when used with relatively long input sequences. Since the longer the interaction region is, the more energy is needed to make it accessible for the interaction. Therefore for longer sequences, accessibility energy is not negligible anymore and has a direct impact on choosing the best interaction. For example consider the interaction given in Figure 1.2 which shows a possible output of *RNAhybrid*.

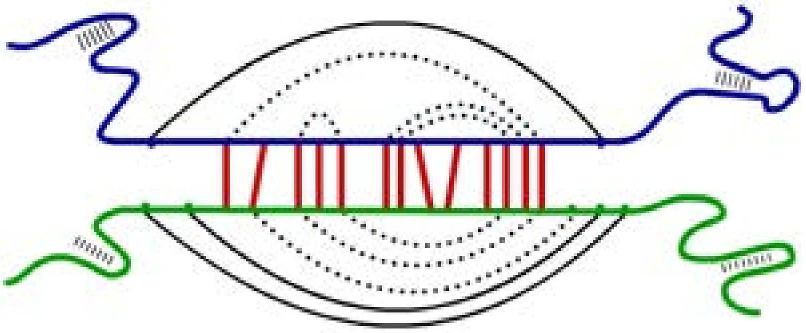


Figure 1.2: The dotted base pairs are the ones required to be broken to make the interacting region free of intramolecular base pairs and available for interaction with the other sequence.

Because the interacting region is long, it may have a good hybridization energy with lots of base pairings between the two sequences. On the other hand, long interaction regions require an amount of accessibility energy which is not easy to overcome. Therefore this structure as an output may not be the biologically correct one.

1.3.3 *IntaRNA*, *RNAup*

IntaRNA and *RNAup* [21] are two other programs for the prediction of RNA-RNA interactions. They overcome all the disadvantages in the previously mentioned algorithms. *IntaRNA* and *RNAup* do not use the concatenation method, and thus allow for loop-loop interactions. Two types of energies contribute to the overall scoring, (1) Hybridization energy (exothermic) resulting from base pairing between the two sequences (2) Accessibility energy (endothermic) required to make the interacting region in each sequence free from intramolecular pairings. The two factors are opposing each other; the more base

pairings we have between the two sequences, the more hybridzation energy is *released* (and thus more stability), in the same time the more accessibility energy is *required.* They balance this tradeoff in a way so that at the end, the best possible interaction is returned as output. They use dynamic programming approach to build up an optimum solution for the resulting interaction. *RNAup* has a time complexity of $O(n^3 + nw^5)$ (where w is the maximum interaction length allowed), which makes it inapplicable on huge datasets, while *IntaRNA* has two versions of the algorithm, a non-heuristic version which is $O(n^2m^2)$, and a heuristic version which is a good approximation for the non-heuristic version, but is much faster with time complexity $O(nm)$ only [9]. The heuristic version is more practical for genome-wide researches due to its time efficiency. However, output accuracy needed to be improved to make the heuristic algorithm more reliable and to widen the set of inputs for which the algorithm give the optimal solution, which is one thing we present in this thesis. Further explanation for the accuracy problems of the heuristic version is found in Section 4.1.1. Another major advantage of *IntaRNA* is that it enables the user to enforce a seed region in the interaction and specify the parameters of this seed.
The drawback of these algorithms is that only one hybridization region is possible in the resulting interaction. An interaction like *OxyS-fhlA* cannot be predicted correctly since they were found experimentially to have two kissing haripins [3]. Even a simpler duplex of multiple interaction sites (as shown in Figure 1.3) cannot be predicted by *IntaRNA* or *RNAup*.

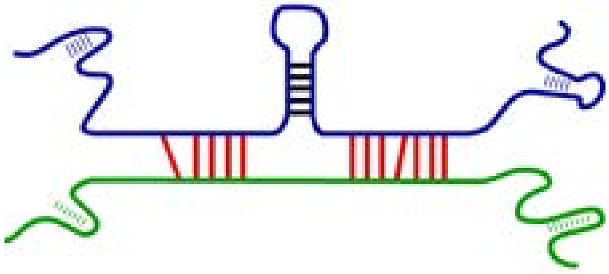


Figure 1.3: A duplex with multiple interaction sites that cannot be predicted by *IntaRNA* or *RNAup*.

1.3.4 *RNArip* , *piRNA*

Those two algorithms [10, 15] can compute joint secondary structures of two interacting RNAs. The computed interaction can contain multiple interaction sites. Specifically *RNArip* was able to predict the *OxyS-fhlA* [3] with good accuracy. It could predict one of the kissing hairpins exactly and predicted the other kissing hairpin with a difference of two base pairs only (compared to the experimentally validated interaction). The problem of those algorithms is the the time requirement of $O(n^6)$. The space complexity for both algorithms is $O(n^4)$.

For more information about different RIP algorithms, see [4] for a review.

1.3.5 Overview

In this thesis we focus on imporving the two algorithms of *IntaRNA*. First, we modify the non-heuristic algorithm in a way that allows to forbid high energy barriers that might be encountered during interaction formation and that are less likely to be overcome, this is discussed in Chapter 3. Second, we improve the heuristic algorithm to make it more accurate and reliable, more details are in Chapter 4. After that we perform an evaluation of both algorithms with respect to prediction accuracy, time consumption and memory consumption. In our experiment we use a data set of 36 pairs of RNAs. We discuss the results in Chapter 5.

Chapter 2

Background

2.1 RNA

RNA (**R**ibo**n**ucleic **A**cid) is a biopolymer composed of a long chain of nucleotides. Each nucleotide is a molecule that is composed of a ribose sugar, a phosphate group and a nitrogenous base. The type of a nucleotide depends on its nitrogenous base. A nucleotide can have one of four types of bases attached to its sugar, these are Adenine, Guanine, Uracil or Cytosine. RNAs play vital roles in the synthesis of proteins that control chemical processes in the cell. An RNA molecule is usually defined by its primary and secondary structures (and possibly other structures which are out of our scope).

Primary structure

The primary structure of an RNA molecule is a sequence of nucleotides that form that molecule. An RNA nucleotide sequence is represented in computational context as a string of characters S over the alphabet {A,G,U,C}. Each character in S represents one nucleotide forming the RNA molecule. Each of the four characters of the alphabet resembles a nucleotide type :

- **A Adenine**
- **G Guanine**
- **U Uracil**
- **C Cytosine**

In general, two nucleotides may form a bond called *base pairing.* A nucleotide can either be free (unpaired) or involved in exactly one base pair at a time, but not more. When an RNA has a region of consecutive free nucleotides, we call it a single-stranded region.

Secondary structure

The secondary structure of an RNA molecule with primary structure S defines the base pairings formed between its nucleotides. It can be formally defined as follows :

$$T = \{(i, i')|\ i \in S,\ \ i' \in S,\ \ S_i \text{ is paired with } S_{i'}\}$$

where T is the set of base pairings over the sequence S and S_i is the ith nucleotide in the sequence S. Each pair (i, i') in the set T represents one base pair between the nucleotides S_i and $S_{i'}$. For the scope of this thesis, we only consider the secondary structures that satisfy the following constraints :

1. $\forall (i, i') \in T,\ \ 1 \leq i < i' \leq |S|$
2. $\forall (i, i'), (j, j') \in T,\ \ i' \neq j$
3. $\forall (i, i'), (j, j') \in T,\ \ i = j \Longleftrightarrow i' = j'$
4. $\neg \exists (i, i'), (j, j') \in T,\ \ i < j < i' < j'$ *(no pseudoknots are allowed).*

In the following, some basic definitions are introduced :

Intramolecular base pairs: Base pairs between nucleotides of a single RNA molecule, the formed structure is called an intramolecular structure (or secondary structure). The process of forming an intramolecular structure is usually refered to by *RNA folding.*

Intermolecular base pairs: Base pairs between nucleotides of two RNA sequences, the formed structure is called an intermolecular structure (RNA-RNA interaction).

Duplex : The words *duplex* and *intermolecular structure* are used interchangeably in this thesis.

Seed region : In intermolecular interactions, a seed region is a region in each of the interacting sequences, which is assumed to initiate the interaction. The seed region consists of consecutive complementary bases in the two sequences that can form a thermodynamically stable initial interaction.

RNA-RNA interactions can be composed of the following secondary structure elements (shown in Figure 2.1) :

Stacking

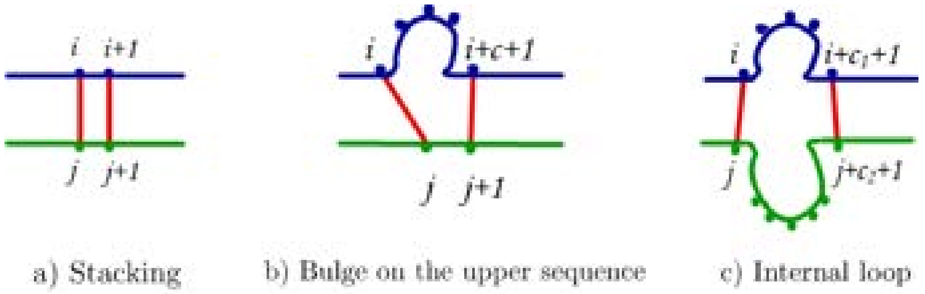
A stacking is formed by two subsequent base pairs having no free nucleotides between them. For example the base pairs (i, j) and $(i + 1, j + 1)$ form a stacking.

Bulge

A bulge is formed by two base pairs when there are one or more unpaired nucleotides between them in *only one* of the two sequences. For example the base pairs (i, j) and $(i+c+1, j+1)$ form a bulge on the first sequence and the base pairs (i, j) and $(i+1, j+c+1)$ form a bulge on the second sequence provided that $c > 0$. c represents the number of unpaired nucleotides between the two base pairs in the corresponding sequence.

Internal loop

An internal loop is formed by two base pairs where there are one or more unpaired nucleotides between them *in both sequences.* For example the base pairs (i, j) and $(i + c_1 + 1, j + c_2 + 1)$ form an internal loop provided that $c_1 > 0$ and $c_2 > 0$. c_1 and c_2 represents the number of unpaired nucleotides between the two base pairs in the first and second sequences respectively.



a) Stacking b) Bulge on the upper sequence c) Internal loop

Figure 2.1: Possible interaction structures

2.2 Energy model of RNA secondary structures

The energy model used for scoring intramolecular and intermolecular structures in this thesis is the *Turner free* energy model.

In this energy model, only the following pairs of nucleotide types are allowed to form a base pair :

- **(G,C)**
- **(A,U)**
- **(G,U)**

Scoring an interaction in *IntaRNA* is based on two energy contributions :

- *Hybridization energy* : Energy score due to intermolecular base pairings in the form of stackings, bulges or internal loops (exothermic energy, i.e., energy is a negative value).
- *Accessibility energy* : An amount of energy required to make the interacting region single-stranded, i.e., not involved in intramolecular pairings (endothermic energy, i.e., energy is a positive value).

The term *ensemble* denotes the set of all secondary structures that can be formed by an RNA sequence.

The *accessibility energy* of a region $[i,k]$ in an RNA sequence S is defined by the energy difference (denoted by ED) :

$$ED(i,k) = -(E^{all} - E^{unpaired}_{i,k}),$$

where E^{all} denotes the energy of the ensemble of all possible secondary structures that can be formed by the sequence S and $E^{unpaired}_{i,k}$ denotes the energy of the ensemble of those structures that have the region $[i,k]$ single stranded. If s is the set of all structures that can be formed by a sequence, then the partition function Z_s is defined as

$$Z_s = \sum_{Q \epsilon s} e^{-\frac{E(Q)}{RT}}$$

where $E(Q)$ is the free energy of the sequence S when folded into the structure Q. The term $e^{-\frac{E(Q)}{RT}}$ denotes *Boltzmann factor.* The partition function is the sum over the Boltzmann factors of all states Q. The energy of the ensemble E^{all} could then be defined as

$$E^{all} = -RT ln(Z_s)$$

The energy of an ensemble of structures is calculated using a partition function approach from [20]. Similarly we can get $E^{unpaired}_{i,k}$ by first calculating the partition function $Z_{s^{unpaired}_{i,k}}$ where $s^{unpaired}_{i,k}$ is the ensemble of all structures that can be formed by the sequence S, with the region $[i,k]$ single stranded. Then $E^{unpaired}_{i,k} = -RT ln(Z_{s^{unpaired}_{i,k}})$, for more details see [21].

Note that accessibility values are not additive, i.e.

$$ED(i,k) \neq ED(i,x) + ED(x,k)$$

2.3 Notations and conventions

To simplify explanation we have to define some conventions that are used in the next chapters :

- S^1 and S^2 refer to the sequences of the two given RNAs.
- S^1_i and S^2_j denote the nucleotides at position i in the sequence S^1 and at position j in the sequence S^2.
- $C(i,j,k,l)$ denotes the energy of the most stable interaction between S^1 and S^2 that begins at i through k in S^1 and j through l in S^2.
- $C(a,b,c,d)$ is a *prefix* of $C(i,j,k,l)$ if each of the following is true :
  $a = i$ $\quad b = j$ $\quad c < k$ $\quad d < l$
- $C(a,b,c,d)$ is a *suffix* of $C(i,j,k,l)$ if each of the following is true :
  $a > i$ $\quad b > j$ $\quad c = k$ $\quad d = l$

- *The number of enclosed bases* in $C(i, j, k, l)$ in S^1 is $k - i + 1$
- *The number of enclosed bases* in $C(i, j, k, l)$ in S^2 is $l - j + 1$
- *The total number of enclosed bases* in $C(i, j, k, l)$ is $(k - i + 1) + (l - j + 1)$
- Since an entry $C(i, j, k, l)$ denotes an interaction, the words *entry* and *interaction* will be used interchangeably.
- s^{loop} denotes the maximum size of an internal loop (the default value is 16).

2.4 Previous IntaRNA algorithms

To be able to present the contribution of this thesis on *IntaRNA*, its previous algorithms [9] are reviewed briefly.

2.4.1 Computing the seed

One of the features of *IntaRNA* is that it enforces a seed region in the resulting interactions. It also allow the user to define the following properties of the seed :

- P : the number of base pairs in the seed region
- b_m^{max} : the maximum number of unpaired bases allowed in the target RNA in the seed region.
- b_s^{max} : the maximum number of unpaired bases allowed in the ncRNA in the seed region.
- b^{max} : the maximum number of unpaired bases allowed in both sequences.

The energy of a seed region is calculated in the same way in all presented algorithms in this thesis using the function $seed.seed(i, j, k, l)$ denotes the best seed interaction (having the four properties set by the user) in the region between (i, j) and (k, l). We store the interaction energies of all seeds from (i, j) in the matrix $seed(i, j, P', b^m, b^s)$, where P' denotes the number of paired bases in S^1 and in S^2; and b^m and b^s denote the number of unpaired bases in S^1 and S^2 repectively. The recursion is as follows :

$$seed(i, j, P', b^m, b^s) =$$

$$\begin{cases} \min\limits_{\substack{p,q \\ i<p\leq min(i+s^{loop}+1,n) \\ j<q\leq min(j+s^{loop}+1,m)}} \begin{cases} \left\{ \begin{array}{l} E^{loop}(i,j,p,q)+ \\ seed(p,q,P'-1, \\ b^m-(p-i-1), b^s-(q-j-1)) \end{array} \right\} & \text{if } P' > 2 \\ E^{loop}(i,j,p,q) & \text{if } P' = 2 \end{cases} & \begin{array}{l}\text{if } (S_i^1, S_j^2) \\ \text{can pair}\end{array} \\ \\ \infty & otherwise \end{cases} \tag{2.1}$$

$E^{loop}(i, j, p, q)$ gives the energy of the loop formed between base pairs (i, j) and (p, q), such that this loop could be any of the interaction structures stacking, bulge or internal loop. Both $seed(i, j, k, l)$ and $seed(i, j, P', b^m, b^s)$ represent the same thing, the equivalence between both representations is clarified in the following equation :

$$seed(i,j,k,l) = \begin{cases} seed(i,j,P,(k-i+1)-P,(l-j+1)-P) & \text{if } k-i+1 \geq P \text{ and} \\ & l-j+1 \geq P \\ \infty & otherwise \end{cases} \tag{2.2}$$

Note the difference between P and P'; P is the number of paired bases in the seed defined by the user, while P' is the number of paired bases in the seed that is denoted by $seed(i, j, P', b^m, b^s)$ with $2 \leq P' \leq P$.

2.4.2 Non-heuristic version

The first algorithm of *IntaRNA* is a complete version, which computes different interactions with all possible starts and ends. The direction of computation is from the right to the left. It is based on dynamic programming with the following recursions :
Initial case :

$$C^{k,l}(k,l) = \begin{cases} ED_1(k,k) + ED_2(l,l) & \text{if } S^1_k, S^2_l \text{ can pair} \\ \infty & otherwise \end{cases} \tag{2.3}$$

Main recursion :

$$C^{k,l}(i,j) = \begin{cases} \min\limits_{\substack{p,q \\ i < p \leq min(i+s^{loop}+1,n) \\ j < q \leq min(j+s^{loop}+1,m)}} \begin{cases} E^{loop}(i,j,p,q) + C^{k,l}(p,q) \\ -ED_1(p,k) - ED_2(q,l) \\ +ED_1(i,k) + ED_2(j,l) \end{cases} & \begin{array}{l}\text{if } (S^1_i, S^2_j) \text{ can pair} \\ \text{and } (S^1_k, S^2_l) \text{ can pair}\end{array} \\ \infty & otherwise \end{cases} \tag{2.4}$$

where $C^{k,l}(i,j)$ denotes the interaction between sequences S^1 and S^2 in the region enclosed by the base pairs (i,j) and (k,l). ED_1 and ED_2 gives the accessibility values for the first and second sequences respectively. $C^{k,l}(i,j)$ is computed by extending another interaction $C^{k,l}(p,q)$ to the left by adding a loop. The extension that gives the minimal energy interaction is taken. Since accessibility values are not additive and accessibility energy is already included in $C^{k,l}(p,q)$, one has to subtract the old accessibility values before adding the new ones.

However, a seed region is not enforced in the interactions of matrix C. For that reason, another matrix $C^{k,l}_{seed}$ stores interactions with a seed. This matrix is computed using the following recursion :

$$C^{k,l}_{seed}(i,j) =$$

$$\begin{cases} \min \left\{ \begin{array}{l} \min\limits_{\substack{p,q \\ i<p\leq min(i+s^{loop}+1,n) \\ j<q\leq min(j+s^{loop}+1,m)}} \begin{cases} E^{loop}(i,j,p,q) + C^{k,l}_{seed}(p,q) \\ -ED_1(p,k) - ED_2(q,l) \\ +ED_1(i,k) + ED_2(j,l) \end{cases} \\ \\ \min\limits_{\substack{p,q \\ P\leq p-i+1\leq P+b^{max}_m \\ P\leq q-j+1\leq P+b^{max}_s \\ (p-i+1)+(q-j+1)\leq 2P+b^{max}}} \begin{cases} seed(i,j,p,q) + C^{k,l}(p,q) \\ -ED_1(p,k) - ED_2(q,l) \\ +ED_1(i,k) + ED_2(j,l) \end{cases} \end{array} \right\} & \substack{\text{if } (S^1_i,S^2_j) \text{ can pair} \\ \text{and } (S^1_k,S^2_l) \text{ can pair}} \\ \infty & otherwise \end{cases} \tag{2.5}$$

Here an entry $C^{k,l}_{seed}(i,j)$ is computed by taking the minimum of the two cases : (1) The entry denotes the extension of interaction that already contains a seed by adding a loop to its left end, (2) The entry denotes an interaction composed of a seed on the left followed by a hybridization stored in C. $seed(i,j,k,l)$ denotes the best seed region between base pairs (i,j) and (k,l).

This version of *IntaRNA* has a time complexity of $O(n^2m^2)$. The space complexity is only $O(nm)$ since the same 2-D matrices $C^{k,l}$ and $C^{k,l}_{seed}$ are reused for all different (k,l).

2.4.3 Heuristic version

The approach of the heuristic version is similar to the complete version but the difference is that for each interaction with a left end base pair (i,j), only one right end (k,l) is considered instead of all possible right ends. This means that the number of possible interactions considered (and consequently the time required) is reduced by a quadratic factor. The recursions follow.

$$C(i,j) =$$

$$\begin{cases} \min \left\{ \begin{array}{ll} \min\limits_{\substack{p,q \\ i<p\leq min(i+s^{loop}+1,n) \\ j<q\leq min(j+s^{loop}+1,m)}} \begin{cases} E^{loop}(i,j,p,q) + C(p,q) \\ -ED_1(p,che_1(p,q)) \\ -ED_2(q,che_2(p,q)) \\ +ED_1(i,che_1(p,q)) \\ +ED_2(j,che_2(p,q)) \end{cases} & (1) \\ \\ ED_1(i,i) + ED_2(j,j) & (2) \end{array} \right\} & \text{if } (S^1_i,S^2_j) \text{ can pair} \\ \infty & otherwise \end{cases} \tag{2.6}$$

$$che(i,j) = \begin{cases} che(p,q) & \text{if Case (1) is the minimum in Equation 2.6} \\ (i,j) & \text{if Case (2) is the minimum in Equation 2.6} \end{cases} \tag{2.7}$$

An entry $C(i,j)$ in the matrix C stores the best interaction with the left end (i,j). Equation 2.6 is similar to Equation 2.4 (complete version recursion) with the difference that here only one right end is considered. $che(i,j)$ (for C Hybridization End) stores the right end for the interaction in $C(i,j)$. Since $che(i,j)$ stores a base pair, $che_1(i,j)$ denotes the first element of the pair and $che_2(i,j)$ denotes the second. In other words, if $che(i,j) = (k,l)$, then $che_1(i,j) = k$ and $che_2(i,j) = l$.

To encorporate a seed interaction, we introduce two additional matrices C^{seed} and che^{seed}. An entry $C^{seed}(i,j)$ in this matrix stores the best interaction with left end (i,j), which includes a seed region. $che^{seed}(i,j)$ stores the right end for the interaction denoted by $C^{seed}(i,j)$. The matrix C^{seed} can be filled according to the following recursion :

$C^{seed}(i,j) =$

$$\begin{cases} \min \left\{ \begin{array}{l} \min\limits_{\substack{p,q \\ i < p \leq min(i+s^{loop}+1,n) \\ j < q \leq min(j+s^{loop}+1,m)}} \begin{cases} E^{loop}(i,j,p,q) + C^{seed}(p,q) \\ -ED_1(p, che_1^{seed}(p,q)) - ED_2(q, che_2^{seed}(p,q)) \\ +ED_1(i, che_1^{seed}(p,q)) + ED_2(j, che_2^{seed}(p,q)) \end{cases} \\ \\ \min\limits_{\substack{p,q \\ P \leq p-i+1 \leq P+b_m^{max} \\ P \leq q-j+1 \leq P+b_s^{max} \\ (p-i+1)+(q-j+1) \leq 2P+b^{max}}} \begin{cases} seed(i,j,p,q) + C(p,q) \\ -ED_1(p, che_1(p,q)) - ED_2(q, che_2(p,q)) \\ +ED_1(i, che_1(p,q)) + ED_2(j, che_2(p,q)) \end{cases} \end{array} \right\} & \text{if } (S_i^1, S_j^2) \text{ can pair} \\ \\ \infty & otherwise \end{cases} \tag{2.8}$$

The first case in Equation 2.8 forms the interaction by extending another interaction - that contains a seed - to the left. The second case forms the interaction from a seed followed by hybridization to its right. The energy of this hybridization is taken from the matrix C.

2.4.4 Dangling end energies

Dangling end energies are additional stabilizing energies which form when an unpaired nucleotide stacks with an adjacent base pair. In both the heuristic and non-heuristic algorithms mentioned above, the dangling end energies are computed first for the rightmost base pair, for which we consider 4 possibilities of dangling ends. After the interaction is extended to the left, the dangling end energies of the leftmost base pair are computed by considering again all 4 possibilities.

2.5 Implementation

ANSI C++ is the programmng language used to implement all the algorithms presented in this thesis. *RNAplfold* [5, 7] and *RNAup* [21] modules (from *ViennaRNA* package [14]) were used to compute the accessibility energy values for the target RNA and ncRNA respectively. The values are stored in a 2-D array for each sequence.

Previously, the E^{loop} function was not implemented. It was only mentioned in theoretical context for recursion simplification. But in the implementation the recursions were expanded to four cases wherever there was an E^{loop} function in a recursion. In this thesis, the E^{loop} function is implemented to make it easy not only for theoretical explanation, but also for implementation and to support code organization and reusability.

Given the four indices of the two base pairs enclosing the loop, the function checks for the type of the structure enclosed between the two base pairs and returns the appropriate energy. The function is shown in Algorithm 1.

Algorithm 1 E^{loop} function

```
function $E^{loop}(i, j, k, l)$

if $k - i = 0$ and $l - j = 0$ then
    return 0
else if $k - i = 1$ and $l - j = 1$ then
    return $stacking(i, j, k, l)$
else if $k - i > 1$ and $l - j = 1$ then
    return $bulge(i, j, k, l)$
else if $k - i = 1$ and $l - j > 1$ then
    return $bulge(i, j, k, l)$
else if $k - i > 1$ and $l - j > 1$ then
    return $internal\_loop(i, j, k, l)$
else
    return $\infty$
end if
end function
```

Chapter 3

Non-heuristic algorithm

3.1 Motivation

Developing a non-heuristic version of IntaRNA is important for several reasons. First, to have an algorithm with accurate output[1], i.e., the optimal solution according to the energy model, so that we can compare to the output of the other heuristic or non-optimal algorithms and produce statistics on accuracy and reliability for them. The second reason, is that in this algorithm, the order we use to calculate the result is similar to the order in which real interactions are assumed to form; that is, interacting first in a seed region of consecutive nucleotides with perfect complementarity then extending the interaction from this seed in both directions (left and right) simultaneously. The order is shown in Figure 3.1.

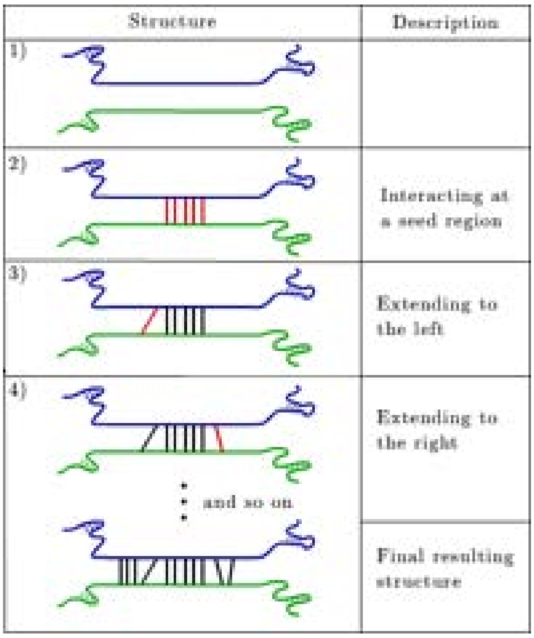


Figure 3.1: The order of computing an MFE interaction structure.

[1] from our point of view, with the restriction of having only one interaction region and using the knowledge available at the time of writing this thesis

This has the major advantage to allow tracking the interaction energy while interaction formation is in process, then stop the interaction at some points where high amounts of energy are required to form further base pairs. For example suppose that the structure in Figure 3.2 is an MFE structure between two sequences.

But to form this structure between the two sequences, the interaction process needs to go from Figure 3.3.a to Figure 3.3.b and then to Figure 3.3.c.

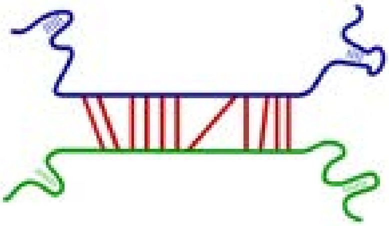


Figure 3.2: MFE structure between two example sequences.

However the intermediate duplex in Figure 3.3.b maybe relatively unstable due to the long bulge on the right end. So it is more probable that the interaction formation stops at the point shown in Figure 3.3.a. This version of IntaRNA is ready to detect such points and stop the interaction when seen appropriate. This feature was not implemented due to the lack of a specific energy threshold, however the algorithm is made ready for the feature in the means of data structures used and code organization.

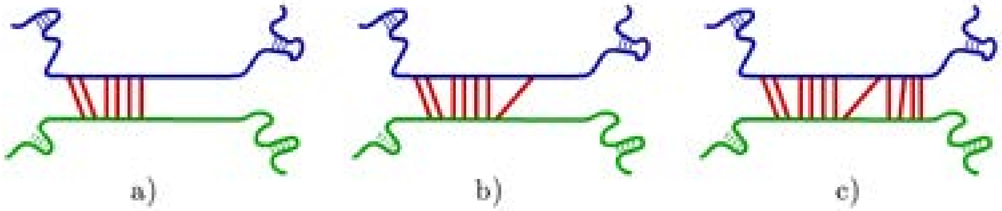


Figure 3.3: Illustration of the energy barrier problem. Duplex b) is relatively unstable due to the long bulge.

3.2 Algorithm

When given two sequences, S^1 and S^2, with lengths n and m respectively, the task is to predict the MFE duplex and its interaction energy value, enforcing a seed region in this duplex. As mentioned before, scoring of an interaction is based on *hybridization energy* and *accessibility energy*. Dangling end energies contribute to the overall energy computed in the end.

The algorithm is based on dynamic programming, and the following is the main recursion used :

$C(i,j,k,l) =$

$$\begin{cases} \min \begin{cases} \displaystyle\min_{\substack{x,y \\ max(k-s^{loop}-1,\,i+P-1)\le x<k \\ max(l-s^{loop}-1,\,j+P-1)\le y<l}} \begin{cases} \boldsymbol{C(i,j,x,y)} + E^{loop}(x,y,k,l) \\ -ED_1(i,x) - ED_2(j,y) \\ +ED_1(i,k) + ED_2(j,l) \end{cases} \\ \\ \displaystyle\min_{\substack{p,q \\ i<p\le min(i+s^{loop}+1,\,k-P+1) \\ j<q\le min(j+s^{loop}+1,\,l-P+1)}} \begin{cases} E^{loop}(i,j,p,q) + \boldsymbol{C(p,q,k,l)} \\ -ED_1(p,k) - ED_2(q,l) \\ +ED_1(i,k) + ED_2(j,l) \end{cases} \\ \\ seed(i,j,k,l) + ED_1(i,k) + ED_2(j,l) \qquad \begin{array}{l} \text{if } P \le k-i+1 \le P+b_m^{max} \\ \text{and } P \le l-j+1 \le P+b_s^{max} \\ \text{and } (k-i+1)+(l-j+1) \le 2P+b^{max} \end{array} \end{cases} & \text{if } S_i^1, S_j^2 \text{ can pair and } S_k^1, S_l^2 \text{ can pair} \\ \infty & otherwise \end{cases} \tag{3.1}$$

$C(i,j,k,l)$ denotes the energy of the best interaction formed between the two sequences S^1 and S^2 in the regions $[i,k]$ in S^1, inclusive, and $[j,l]$ in S^2 , inclusive too. Thus, since $[i,k]$ denotes a region in S^1 beginning at i and ending at k, it is always ensured that $k > i$, and the same for j and l in S^2, in order for the entry $C(i,j,k,l)$ to be valid.

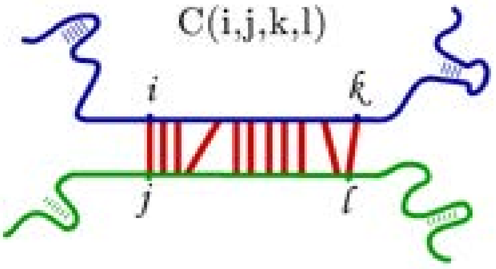


Figure 3.4: An interaction as stored in the matrix entry $C(i,j,k,l)$

In the recursion, the hybridization energy terms are $E^{loop}(x,y,k,l)$ and $E^{loop}(p,q,k,l)$. $E^{loop}(x,y,k,l)$ is the free energy of the loop beginning at the base pair (i,j) and ending at base pair (k,l). The maximum size of a loop is restricted to s^{loop} with default value of 16, i.e. $k-x-1 \le 16$ and $l-y-1 \le 16$. Accessibility energy terms are ED_1 for accessibility of regions in S^1 and ED_2 for accessibility of regions in S^2. $ED_1(i,k)$ denotes the energy required to make all the nucleotides i to k - inclusive - accessible and not involved in intramolecular structures.

The idea of the recursion is clarified in Figure 3.5. The entry $C(i,j,k,l)$ is the minimum of three cases. Case (A) refers to the case when $C(i,j,k,l)$ is an extension with a loop to the right of another interaction in $C(i,j,x,y)$, the minimum is taken over all possible (x, y), and thus over all possible interaction extensions to the right. Similarly Case (B) is the minimum energy value of extending an interaction by adding a loop to its left side. Case (C) refers to the case when $C(i, j, k, l)$ is a seed, where $seed(i, j, k, l)$ gives the energy of the best seed in the region between base pairs (i, j) and(k, l) as mentioned previously in Section 2.4.1.

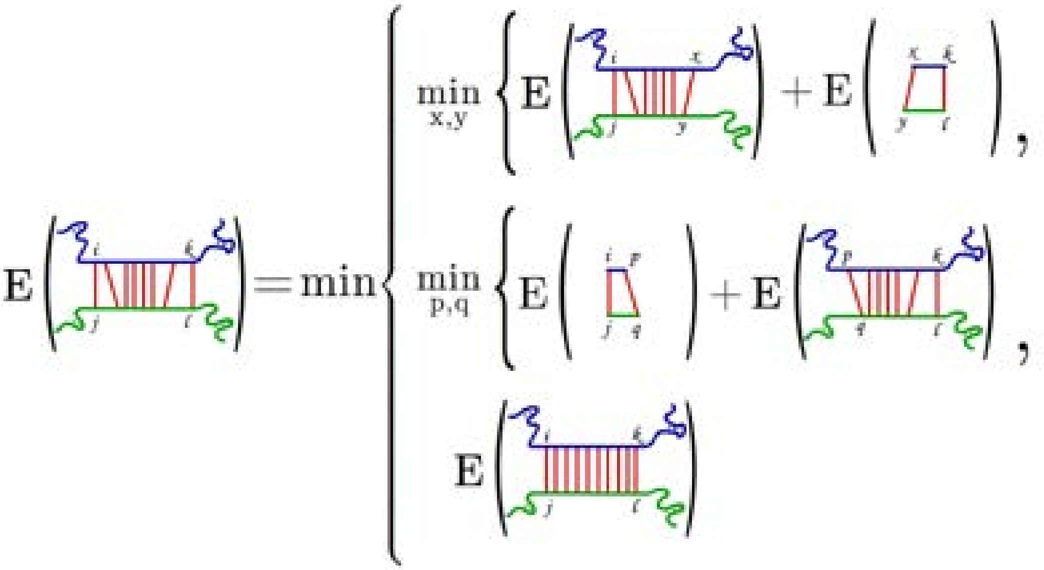


Figure 3.5: Visual representation of recursion in Equation 3.1.

Accessibility energies are not additive. When the matrix entry $C(i, j, k, l)$ stores the best interaction from (i, j) to (k, l), for example if it was computed from $C(i, j, x, y) + E^{loop}(x, y, k, l)$. We first have to subtract the accessibility energy values $ED_1(i, x)$ and $ED_2(j, y)$ which are already included in $C(i, j, x, y)$, then add the accessibility values $ED_1(i, k)$ and $ED_2(j, l)$ to $C(i, j, k, l)$.

In this context, we can say that the order in which a duplex energy is computed is the same order in which a real duplex forms. The two sequences first interact in a seed region with almost perfect complementarity forming the basis of the interaction due to its thermodynamic stability, then this seed is extended to the left and the right simultaneously. To clarify more, the duplex in Figure 3.4 may be computed or formed in the order shown in Figure 3.1.

After all entries of the matrix C are filled in the correct order, dangling end energies are added to each entry :

$$C^{dangle}(i, j, k, l) = C(i, j, k, l) + dangling(i, j, k, l)$$

The function $dangling(i, j, k, l)$ computes the best scoring dangling ends for the interaction in $C(i, j, k, l)$. For each of the four bases enclosing the interaction i, j, k, l, a dangling end, i.e., an unpaired nucleotide, can be included or excluded at positions $i - 1$ and $j - 1$ left of i and j, respectively, and at positions $k + 1$ and $l + 1$ right of k and l, respectively. Having 2 possibilities for each of the 4 bases enclosing the interaction leaves us with $2^4 = 16$ combinations of dangling end energies. The minimum of the 16 values is taken. The bases included as dangling ends are also considered in the accessibility energy. For example if bases $i-1$ and $k+1$ were included as dangling ends and bases $j-1$ and $l+1$ were excluded , the accessibility energy of $C(i,j,k,l)$ is subtracted, and instead we add $ED_1(i-1,k+1) + ED_2(j,l)$. Note that the way we compute dangling end energies here is different from the previous *IntaRNA* non-heuristic version. Here we compute the dangling ends after we fill the matrix (i.e. after the interaction is formed) and we consider 16 cases, while in the previous version, danling ends energy was computed for the right most base pair first then for the left most base pair after the interaction has been extended (see Section 2.4.4).

Finally, the matrix C is searched to find the MFE interaction :

$$MFE = \min_{i,j,k,l} \left\{ \quad C^{dangle}(i,j,k,l) \quad \right\}$$

After the entry of the minimum energy was found, a traceback on matrix C is done to get the MFE structure.

Since the loop size is restricted to constant number (default is 16), the time complexity of the algorithm is $O(n^2m^2)$. The space complexity is $O(n^2m^2)$ too.

3.3 Implementation

In this section we explain the implementation details of the algorithm.

3.3.1 The order of filling the matrix

For $C(i,j,k,l)$ we use a 4-D array. Initially all entries are set to ∞. Then the matrix is filled according to the recursion 3.1. The order of filling the matrix cannot be simply done using 4 nested loops as shown in Algorithm 2.

Algorithm 2 Normal order of filling a matrix

```
for $i = 1$ to $n$ do
  for $j = 1$ to $m$ do
    for $k = i$ to $n$ do
      for $l = j$ to $m$ do
        Fill $C(i,j,k,l)$ using the recursion 3.1
      end for
    end for
  end for
end for
```

n and m are the lengths of S^1 and S^2 respectively. The problem is that computation of some entries relies on other entries that have not yet been computed. For example, the entry $C(1,1,8,8)$ is computed before the entry $C(2,2,8,8)$ which should not be allowed according to our recursion because $C(1,1,8,8)$ uses $C(2,2,8,8)$. One could solve this particular dependency by changing the for loops order to be descending. But then the problem would be with entries like $C(1,1,7,7)$ which are also needed by $C(1,1,8,8)$ but filled after it.

For that reason another approach was taken to solve this problem. To fill the matrix correctly we should mantain the invariant that when an entry is to be computed, all its *prefix* and *suffix*[1] entries have to be already computed. Since for an entry $C(i,j,k,l)$ all its *prefix* and *suffix* entries enclose a smaller number of bases in S^1 and a smaller of bases in S^2, therefore the *total number of enclosed bases* is also smaller. Let *blen* denote the *total number of enclosed bases*. Thus we can first loop on *blen* ascendingly, ensuring that entries with a smaller *total number of enclosed bases* are computed and filled first. The 4 nested loops can be written as follows :

Algorithm 3 Correct order of filling C

```
 1: for $blen = 2$ to $(n+m)$ do
 2:   for $i = 1$ to $n$ do
 3:     for $j = 1$ to $m$ do
 4:       for $len1 = 1$ to $min(blen-1, n-i+1)$ do
 5:         $len2 = blen - len1$
 6:         $k = i + len1 - 1$
 7:         $l = j + len2 - 1$
 8:         Fill $C(i,j,k,l)$ using the recursion 3.1
 9:       end for
10:     end for
11:   end for
12: end for
```

The second and the third for loop get all possible left base pairs (i,j) of the interaction, the fourth for loop iterates over the number of enclosed bases in S^1 ($len1$). Then the number of enclosed bases in S^2 ($len2$) is derived in Line 5. Now $len1$ is the number of enclosed bases in S^1 including i, and $len2$ is the number of enclosed bases in S^2 including j. Consequently, $k = i + len1 - 1$, $l = j + len2 - 1$. Additionally $len1$ is enforced to be less than or equal *blen* -1, so that $len2$ is at least equal to 1, and it is enforced to be less than $n - i + 1$ so that k does not exceed the sequence length n. Having now the four indices ready, the corresponding matrix entry can be filled according to recursion 3.1. However, leaving the loops in this way can lead at some points to invalid indices. Thus, before filling the entry, one should make sure that the four indices represent a valid entry. A quadraple (i,j,k,l) that represents a valid entry should satisfy all the following properties :

1. $1 \leq i \leq n$

2. $1 \leq j \leq m$

3. $1 \leq k$

[1] For a definition of *prefix* and *suffix* entries refer to Section 2.3

4. $1 \leq l$

5. $k \leq n$

6. $l \leq m$

7. $k - i + 1 \geq P$ (since $P \geq 1$, this rule also implies : $k > i$)

8. $l - j + 1 \geq P$ (since $P \geq 1$, this rule also implies : $l > j$)

Properties 1 and 2 are already enforced by the conditions of the second and third for loops. Properties 3 and 4 are also enforced since $k = i + len1 - 1$, and since $i \geq 1$ and $len1 \geq 1$, similarly for l. Property 5 is enforced in the inner most for loop since $len1$ cannot exceed $n - i + 1$. Yet, Properties 6 to 8 are not enforced by the code written in Algorithm 3. Properties 7 and 8 ensure that all entries that will be filled contain an interaction with a seed; so that all entries without a seed remain ∞. In other words they ensure that *the number of enclosed bases* in S^1 and S^2 are at least P for each sequence. This implies that *the total number of enclosed bases* ($blen$) should be at least $2 * P$. $len1$ also should be in the range [P to $(blen - P)$] so that it allows S^1 to cover a minimum of P base pairs and also does not exceed $(blen - P)$ so that $len2$ is at least P. Property 6 can be enforced by an if statement. This leads us to the following code, where all changes compared to Algorithm 4 are highlighted in red :

Algorithm 4 Correct order of filling C

```
for $blen = 2 * P$ to $(n + m)$ do
  for $i = 1$ to $n$ do
    for $j = 1$ to $m$ do
      for $len1 = P$ to $min(blen - P, n - i + 1)$ do
        $len2 = blen - len1$
        $k = i + len1 - 1$
        $l = j + len2 - 1$
        if $(l \leq m)$ then
          Fill $C(i, j, k, l)$ using the recursion 3.1
        end if
      end for
    end for
  end for
end for
```

Although this code works fine, unuseful iterations of the second and third for loops should be avoided due to efficiency reasons. Since properties 5 and 7 are already enforced, all iterations with $i > n - P + 1$ are redundant, because even when k is maximum (when $k = n$), S^1 would not fit a seed. For example if $i = n - P + 2$ and $k = n$, the number of enclosed bases in S^1 is $k - i + 1 = n - (n - P + 2) + 1 = n - n + P - 2 + 1 = P - 1 < P$. The same applies to j and S^2. Thus the code could be written as follows, with all changes highlighted in red :

Algorithm 5 Correct order of filling C

```
for $blen = 2 * P$ to $(n + m)$ do
  for $i = 1$ to $(n - P + 1)$ do
    for $j = 1$ to $(m - P + 1)$ do
      for $len1 = P$ to $min(blen - P, n - i + 1)$ do
        $len2 = blen - len1$
        $k = i + len1 - 1$
        $l = j + len2 - 1$
        if $(l \leq m)$ then
          Fill $C(i, j, k, l)$ using the recursion 3.1
        end if
      end for
    end for
  end for
end for
```

Now most of the unuseful iterations are avoided, however a few more conditions are applied to the second, third and fourth *for* loops to avoid all iterations that give invalid entries. Those conditions are not mentioned here for simplicity.

3.3.2 Filling an entry in the matrix

To fill an entry of the matrix $C(i, j, k, l)$, we first make sure that (i, j) can pair and (k, l) can pair, otherwise we set the entry to ∞ and exit the function. If they can pair, then we directly compute the energies for the three cases of the recursion and fill the entry with their minimum. The function that fills an entry is given in Algorithm 6.

The first and second case (where $C(i, j, k, l)$ is an extension to left or right of another interaction) are only applicable when the entry covers enough base pairs to fit at least a seed plus a *loop* (stacking, bulge or internal loop); otherwise only the third case (*seed*) is applicable. This is ensured in Line 8 by the *if* statement. For the first case of the recursion (from Line 10) x and y are chosen such that :

1. The size of the loop $E^{loop}(x, y, k, l)$ does not exceed the maximum loop size(s^{loop}).

2. The interaction to be extended $C(i, j, x, y)$ can at least hold a seed.

3. The loop size is at least 0 ($E^{loop}(k - 1, l - 1, k, l)$), which represents a stacking.

Then the minimum energy resulting from all possible extensions (or all possible (x, y)) is chosen for Case 1. Case 2 is similar to Case 1 by symmetry. Case 3 is the case where $C(i, j, k, l)$ is a seed.

Algorithm 6 Filling an entry of C

```
function $fill\_C(i, j, k, l)$

if $CannotPair(i, j)$ $or$ $CannotPair(k, l)$ then
    $C[i][j][k][l] = \infty$
    return
end if

if $k - i + 1 > P$ $and$ $l - j + 1 > P$ then

    //Case 1
    minRight=$\infty$
    for $x = max(k - s^{loop} - 1, i + P - 1)$ to $k - 1$ do
        for $y = max(l - s^{loop} - 1, j + P - 1)$ to $l - 1$ do
            E=$C[i][j][x][y] + E^{loop}(x, y, k, l) - ED_1(i, x) - ED_2(j, y) + ED_1(i, k) + ED_2(j, l)$
            minRight=$min$(minRight ,E)
        end for
    end for

    //Case 2
    minLeft=$\infty$
    for $p = i + 1$ to $min(i + s^{loop} + 1, k - P + 1)$ do
        for $q = j + 1$ to $min(j + s^{loop} + 1, l - P - 1)$ do
            E=$E^{loop}(i, j, p, q) + C[p][q][k][l] - ED_1(p, k) - ED_2(q, l) + ED_1(i, k) + ED_2(j, l)$
            minLeft=$min$(minLeft , E)
        end for
    end for

end if

// Case 3
seedEnergy=$\infty$
if $P \leq k - i + 1 \leq P + b_m^{max}$ $and$ $P \leq l - j + 1 \leq P + b_s^{max}$
 $and$ $(k - i + 1) + (l - j + 1) \leq 2P + b^{max}$ then
    seedEnergy=$seed(i, j, k, l) + ED_1(i, k) + ED_2(j, l)$
end if

$C[i][j][k][l] = min(minRight, minLeft, seedEnergy)$
end function
```

3.3.3 Space improvement

As mentioned before, the space complexity of the algorithm is $O(n^2m^2)$. The space is predominantly taken by the matrix C, which needs n^2m^2 entries. Since we use a data type $float$ for each entry, the space required is approximately $4 * n^2m^2$ bytes (neglecting the space required for accessibility matrices and other variables).
This means that for an input with $n = 250$ and $m = 250$, the space required is :

$$\frac{4 * 250^2 * 250^2}{2^{30}}\, GB \approx 14.55\, GB$$

which is not even feasibly by a modern personal computer nowadays.
Although this algorithm is meant to be a complete version of *IntaRNA* and it was already expected to require much resources, we worked on reducing the required space to make the algorithm applicable for longer inputs.

Since an entry of the matrix C has to be valid in the context of interactions, many entries that do not conform to those rules are not valid and redundant. Strictly speaking, those entries with $k < i$ or $l < j$ are redundant.

The first idea was to use hash tables with a key as an encoding of a 4-tuple (i, j, k, l). This idea was implemented and tested. The hash table reduced the space drastically, because only entries conforming the rules and for which (i, j) or (k, l) can pair were stored. But in the same time using a hash table had a big impact on the speed efficiency. Additionally it put a limit on the input sequence lengths because the key data type was *unsigned long integer*. Consequently, the possible keys were limited to 2^{32}, which limited the sequence lengths to $2^8 = 256$ each.

A small experiment was done to compare time and space requirements when using a 4-D array and a hash table. The input used for the experiment had lenghts $n = 175$ and $m = 60$. The experiment was conducted on a machine with 2.2 GHz processor and 4 GB of RAM. The results are shown in Figure 3.6; using a hash table achieved low memory consumption but long execution time, while using a 4-D array had high memory consumption and a shorter execution time.

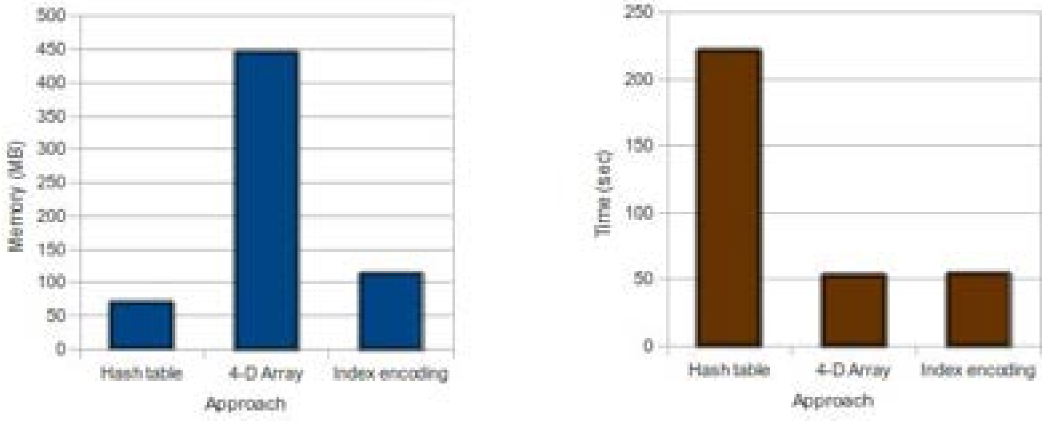


Figure 3.6: Resource requirements of implementations using a hash table, a 4-D array or index encoding for one example

The idea of the hash table was excluded due to the sequence length limit and the time requirement.
However another idea was employed using an array to keep the speed efficiency and decrease the space requirement.

The basic idea is to use an array with smaller size to avoid the invalid entries with $k < i$ or $l < j$. To achieve this, we use a 2-D array such that the first index is an encoding for the two indices i and k and the second index is an encoding for j and l. We denote the first index ik and the second one jl, where each defines a range in one input sequence. This approach allows to encode only the valid indices and exclude the invalid ones.

To explain, first consider this encoding idea without reducing the space (having the same number of entries, but in a 2-D array instead of a 4-D array).

ik (Encoding)	i	k
0	0	0
1	0	1
2	0	2
3	0	3
4	1	0
5	1	1
6	1	2
7	1	3
8	2	0
9	2	1
10	2	2
11	2	3
12	3	0
13	3	1
14	3	2
15	3	3

ik (Encoding)	i	k
0	0	0
1	0	1
2	0	2
3	0	3
4	1	1
5	1	2
6	1	3
7	2	2
8	2	3
9	3	3
10		
11		
12		
13		
14		
15		

Table 3.1: a) All possible (i, k). b) Unuseful rows removed.

Table 3.1.a shows the indices i and k and their encoding ik when $n = 4$ as an example. The total entries for all possible (i, k) is $n * n = 4 * 4 = 16$, so the encoding index range is 0 to 15 assuming that all indices are zero based. For the encoding of 2 indices to 1 index, the mapping function is :

$$map(i, k) = ik = i * n + k$$

Note that in Table 3.1.a the rows highlighted in red should be excluded because $k < i$. All the rows highlighted in blue will be shifted upwards in the table to get the new encoding. Please note that the first i many rows are excluded from all rows with index i. The new encoding is shown in Table 3.1.b.

The encoding function for the reduced space table is :

$$map'(i, k) = ik = i * n + k - \sum_{x=0}^{i} x = i * n + k - \frac{i(i+1)}{2}$$

The summation term subtracted in the map' function is to shift upwards a row from its previous position in the map function by the number of rows to be excluded above this row. The number of excluded rows above the current row equals to $\sum_{x=0}^{i} x$ because for all rows with index i, we have to exclude i rows.

By inspecting Table 3.1.b, we find that the total number of entries is reduced to $\sum_{x=1}^{n} x = \frac{n(n+1)}{2}$

For large n this is almost half the number of entries in the first table, which was n^2. Encoding the two indices j and l in the same way, we end up with $\frac{n(n+1)}{2}$ and $\frac{m(m+1)}{2}$ entries for the first and second dimension, respectively, of the 2-D array. The total size of the array used for storing all the entries is then equal to :

$$\frac{n(n+1)}{2} * \frac{m(m+1)}{2} = \frac{(n^2+n)*(m^2+m)}{4}$$

Algorithm 6 from Section 3.3.2 does not have to be changed because the space reduction is totally abstracted from the algorithm itself by the two functions $C_{write}(i, j, k, l, value)$ and $C_{read}(i, j, k, l)$ that allow writing and reading to the matrix using the four indices. These two functions encode the four indices using the map' function and perform the appropriate operation.

After applying this idea, the space required was almost 4 times less. But there was an overhead because of the extensive use of the mapping functions for each read or write operation. To overcome this speed overhead, all the mappings were computed in a preprocessing stage and stored in two 1-D arrays, which required $\frac{n(n+1)}{2} + \frac{m(m+1)}{2}$ space.

An experiment was conducted again to compare the new approach to the hash tables and the 4-D array. The results of the index encoding are shown in the corresponding columns in Figure 3.6.

Now, an input with lengths $n = 250$ and $m = 250$ needs only 3.66 GB compared to 14.55 GB when using a 4-D array.

3.3.4 Limiting interaction length

The algorithm optionally allows the user to define the maximum number of bases involved (i.e. *number of enclosed bases*) in the predicted interaction. We denote the user defined interaction length by L. When the interaction length is limited, the program need not form or store longer interactions, and thus the required memory and time are reduced.

For reducing the memory, only the mapping function and the size of the used 2-D array have to be modified. For the mapping function, the idea is the same as in Section 3.3.3. Some rows with $k - i + 1 > L$ are excluded from Table 3.1.b and the subsequent rows are shifted upwards by subtracting another term from the previous mapping function. The mapping function of indices i and k is shown in Algorithm 7.

Algorithm 7 Mapping function

```
function $map''(i, k)$
$ik = i * n + k - \sum_{x=1}^{i} x$
if the user $Restricts$ interaction length then
    $ik = ik - \sum_{x=n-L-(i-1)}^{n-L} x$
end if
return $ik$
end function
```

The lengths of the 2-D array dimensions are $(n * L) - \frac{L(L-1)}{2}$ and $(m * L) - \frac{L(L-1)}{2}$, respectively.

Reducing the time required is done by considering only entries $C(i, j, k, l)$ with $k - i + 1 \leq L$ and $l - j + 1 \leq L$. This is done by modifying the for loops in Algorithm 5 in Section 3.3.1. Modifications are highlighted in red in Algorithm 8.

The reduction in time and memory requirements are shown in the graph in Figure 3.7, it shows the results when *IntaRNA* was executed with varying interaction length limit (L) on the RNAs *SgrS* and *ptsG* from the bacterial organism *Escherichia coli.* The lengths of those two RNAs are 227 and 250, respectively. The length of the predicted interaction between both RNAs is 20.

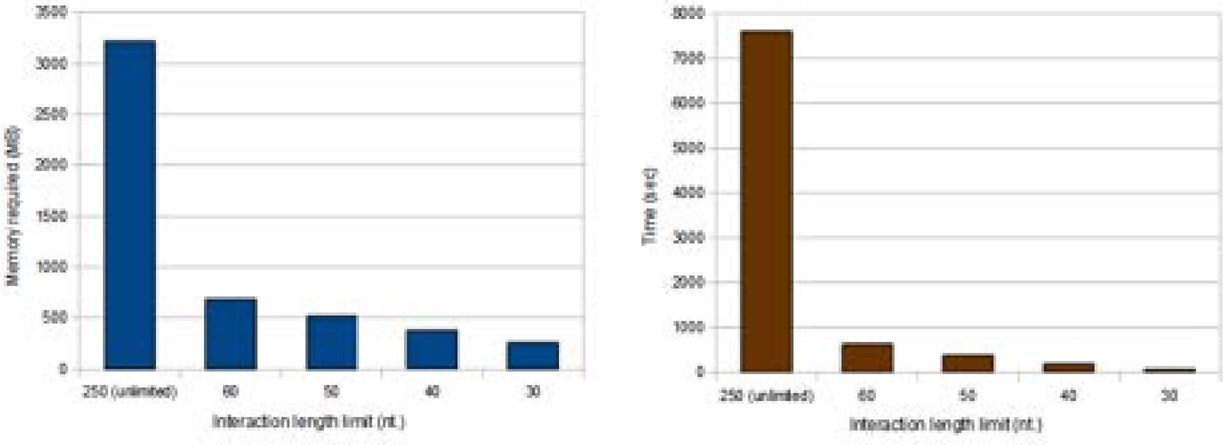


Figure 3.7: A graph showing time and memory results for executions of non-heuristic algorithm with varying interaction length limits.

Algorithm 8 Correct order of filling C

```
if the user does not restrict interaction length then
  $L = n + m$
end if
for $blen = 2 * P$ to $min((n + m), L)$ do
  for $i = 1$ to $(n - P + 1)$ do
    for $j = 1$ to $(m - P + 1)$ do
      for $len1 = P$ to $min(blen - P,\ n - i + 1,\ L)$ do
        $len2 = blen - len1$
        $k = i + len1 - 1$
        $l = j + len2 - 1$
        if $(l \leq m$ and $len2 \leq L)$ then
          Fill $C(i, j, k, l)$ using the recursion 3.1
        end if
      end for
    end for
  end for
end for
```

Chapter 4

Improved heurestic algorithm

4.1 Motivation

The previous *IntaRNA* heuristic algorithm was a fast variant of the non-heuristic version. It has time complexity of $O(nm)$ because it considers for each interaction left end only one right end instead of all ends[1] that were considered by non-heuristic version. In consequence, for some inputs the algorithm may miss the MFE structure. Thus, an improvement was needed to increase the accuracy of this version of *IntaRNA*. To illustrate the problem, two examples are given in the next section.

4.1.1 Problems of heuristic version

Since the heuristic version extends an interaction to the left without considering all possible right ends, it might sometimes be useful to choose a different right end accordingly as it extends to the left. Figure 4.1 shows a structure that, when extended to the left, is more favorable when being extended to the right as well. Extending the interaction by one base pair to the left breaks an intramolecular base pair (in the upper RNA sequence).

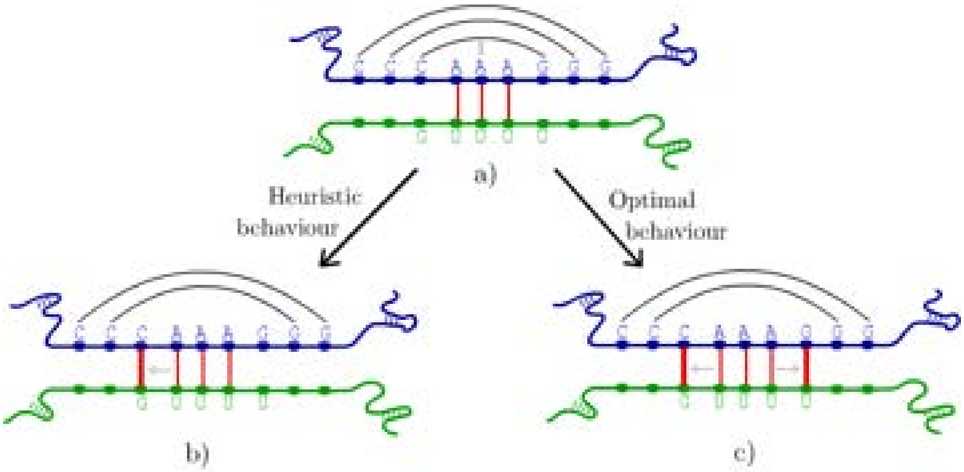


Figure 4.1: Comparing heuristic and optimal behaviors for an example interaction.

[1] For each left end the number of right ends is proportional to $n * m$

Another base pair to right could be added with nearly no extra energy penalty caused by accessibility. However, the heuristic algorithm does not accommodate this.

Another problem is illustrated in Figure 4.2. If the interaction in Figure 4.2.a is extended by one base pair to the left, then intramolecular base pair 1 will be broken. Then, it might be better to allow intramolecular base pair 2 to be formed instead to avoid a high energy penalty due to accessibility. Allowing base pair 2 to form means breaking the intermolecular base pair on the right end of the interaction (base pair 3), which is not possible with the current heuristic algorithm. Instead, the algorithm breaks both base pair 1 and 2, which requires an unfolding energy that might be be greater than the hybridization energy gained from forming base pairs 3 and 4.

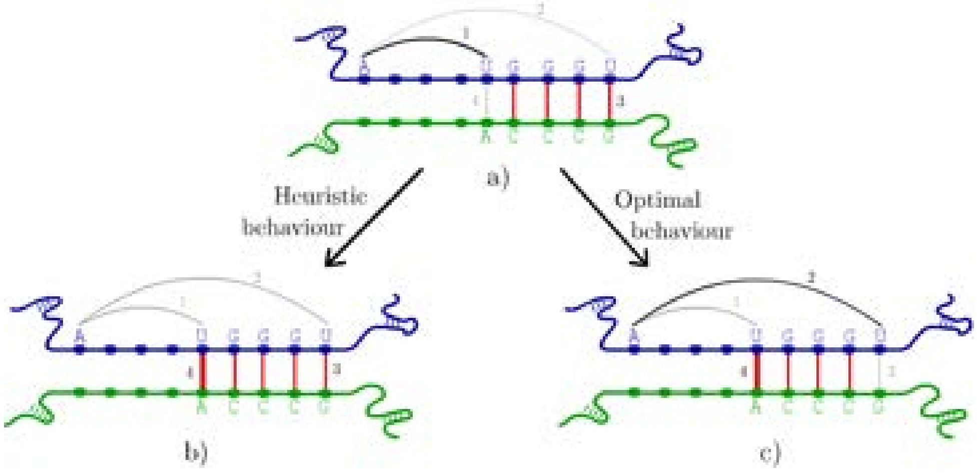


Figure 4.2: Comparing heuristic and optimal behaviors for an example interaction.

In the Figures 4.1 and 4.2 the heuristic algorithm behavior is shown and compared to the behavior of an optimal algorithm.

4.2 Algorithm

The general outline of the algorithm of the improved heuristic version is similar to the one for the previous heuristic version, which was explained in Section 2.4.3.
To tackle the problems mentioned in the previous section, a compromise was made between both the non-heuristic and the heuristic versions. For each left end (i, j), the improved heuristic version stores the energy of the V best interactions with V different right ends, instead of storing the energy for only one right end as in the previous heuristic version, and instead of storing all possible right ends as in the non-heuristic version. The number V is a parameter which can be defined by the user. Thus now $C(i, j)$ could be seen as an array of energy values of size V. An element in this array is denoted by $C(i, j, K)$. $C(i, j, K)$ denotes the energy of the interaction with left end (i, j) having the K^{th} *best right end*. The right interaction end itself is stored in $che(i, j, K)$ (illustrated in Figure 4.3). For a pair $che(i, j, K)$, the sequence positions for the first and second sequence are

denoted by $che_1(i,j,K)$ and $che_2(i,j,K)$, respectively. K^{th} *best right end* denotes the right end that gives the K^{th} minimum interaction energy. We use the following notation

$$\overset{K^{\mathbf{th}}}{\underset{x}{\min}}\{x\},$$

to denote the K^{th} minimum value of all x. The recursion for the 3-D matrix C is shown in Equation 4.1. The recursion is similar to Equation 2.6 (the recusion of the heuristic version, which is explained in Section 2.4.3) with the differences mentioned above.

$$C(i,j,K) =$$

$$\begin{cases} \overset{K^{\mathbf{th}}}{\underset{\substack{p,q,R \\ i<p\leq min(i+s^{loop}+1,n) \\ j<q\leq min(j+s^{loop}+1,m) \\ 1\leq R\leq V}}{\min}} \left\{ \begin{array}{ll} E^{loop}(i,j,p,q)+C(p,q,R) & \\ -ED_1(p,che_1(p,q,R)) & \\ -ED_2(q,che_2(p,q,R)) & (1) \\ +ED_1(i,che_1(p,q,R)) & \\ +ED_2(j,che_2(p,q,R)) & \\ & \\ ED_1(i,i)+ED_2(j,j) & (2) \end{array} \right\} & \text{if } (S^1_i,S^2_j) \text{ can pair} \\ \infty & otherwise \end{cases} \tag{4.1}$$

$$che(i,j,K) = \begin{cases} che(p,q,R) & \text{if Case (1) has the } K^{th} \text{ minimum in Equation 4.1} \\ (i,j) & \text{if Case (2) is the } K^{th} \text{ minimum in Equation 4.1} \end{cases} \tag{4.2}$$

Since the direction of interaction extension is from the right to the left, to fill the array $C(i,j)$ the algorithm considers extending all possible interactions with left end (p,q). For each (p,q) we consider the V different right ends, in other words we consider all possible $C(p,q,R)$ with $1 \leq R \leq V$. Additionally we consider the case when (i,j) is the right most base pair (i.e. first base pair) of a new interaction. From all these possibilities the minimum V energy values are taken and assigned to the array denoted by $C(i,j)$, such that $C(i,j,K)$ corresponds to the K^{th} minimum energy found. It is ensured that for each left end (i,j) we store the V minimum values for *different* right ends, so for two equal right ends, the one with lower energy is taken. The reason for this should be clear when we review the non-heuristic version, which stores for each (i,j,k,l) only one energy value. It can be concluded that the improved heuristic algorithm takes at each step not only the optimal right end, but it also takes another $V-1$ suboptimal right ends. One of those suboptimal right ends could eventually lead to a better result than the optimal right end as shown in in the previous section.

To include a seed region we have to use two additional matrices C^{seed} and che^{seed} for which the recursions follow.

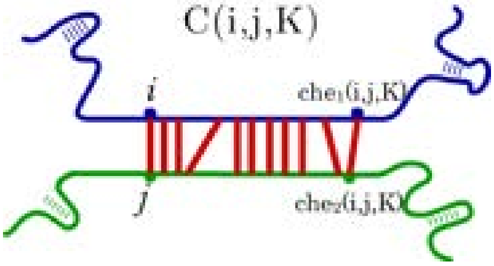


Figure 4.3: An interaction represented by $C(i,j,K)$

$$C^{seed}(i,j,K) =$$

$$\begin{cases} \min\limits^{K^{\mathbf{th}}} \left\{ \begin{array}{ll} \min\limits^{K}_{\substack{p,q,R \\ i<p\le min(i+s^{loop}+1,n) \\ j<q\le min(j+s^{loop}+1,m) \\ 1\le R\le V}} \begin{cases} E^{loop}(i,j,p,q)+C^{seed}(p,q,R) \\ -ED_1(p,che_1(p,q,R)) \\ -ED_2(q,che_2(p,q,R)) \\ +ED_1(i,che_1(p,q,R)) \\ +ED_2(j,che_2(p,q,R)) \end{cases} & (1) \\ \min\limits^{K}_{\substack{p,q,R \\ P\le p-i+1\le P+b_m^{max} \\ P\le q-j+1\le P+b_s^{max} \\ (p-i+1)+(q-j+1)\le 2P+b^{max} \\ 1\le R\le V}} \begin{cases} seed(i,j,p,q)+C(p,q,R) \\ -ED_1(p,che_1(p,q,R)) \\ -ED_2(q,che_2(p,q,R)) \\ +ED_1(i,che_1(p,q,R)) \\ +ED_2(j,che_2(p,q,R)) \end{cases} & (2) \end{array} \right\} & \text{if } (S_i^1,S_j^2) \text{ can pair} \\ \infty & otherwise \end{cases} \tag{4.3}$$

$$che^{seed}(i,j,K) = \begin{cases} che^{seed}(p,q,R) & \text{if Case (1) has the } K^{th} \text{ minimum in Equation 4.3} \\ che(p,q,R) & \text{if Case (2) has the } K^{th} \text{ minimum in Equation 4.3} \end{cases} \tag{4.4}$$

Equation 4.3 is analogous to Equation 2.8. Here, with the same concept, $C^{seed}(i,j)$ is an array of V values with an element denoted by $C^{seed}(i,j,K)$. $C^{seed}(i,j,K)$ denotes the energy of the interaction with left end (i,j) having the *Kth best right end* and including a seed region. The right end base pair positions of the interaction denoted by $C^{seed}(i,j,K)$ are stored in $che^{seed}(i,j,K)$. $C^{seed}(i,j,K)$ is computed by taking the Kth minimum energy value from the two sets resulting from case (1) and case (2), in Equation 4.3, respectively. Case (1) extends another interaction with left end (i,j) that already contains a seed by adding a loop on the left. For each left end (i,j), V possible right ends are considered. Case (2) introduces a seed region with left end (i,j) followed by hybridization. Again for each left end (i, j), V right ends are considered. We used the following notation :

$$\overset{K}{\underset{x}{\min}} \{x\},$$

to denote *the set of the K minimum values.* Please note the difference between the two notations $\overset{K}{\underset{x}{\min}} \{x\}$ and $\overset{K^{\mathbf{th}}}{\underset{x}{\min}} \{x\}$, where the first denotes a set and the second denotes a single value. To fill an entry $C^{seed}(i, j, K)$ we need to get the K^{th} minimum value from the union of the two sets in case (1) and case (2). First we get the K minimum values from each set, then from the union of the two resulting subsets, we get the K^{th} minimum value.

We fill the matrix C followed by C^{seed}, then we add the dangling end energies the same way it was done for the non-heuristic version (see Section 3.2) :

$$C^{seed}_{dangle}(i, j, K) = C^{seed}(i, j, K) + dangling(i, j, che_1(i, j, K), che_2(i, j, K))$$

Note that we compute the dangling end energies different from the previous *IntaRNA* heuristic version (see Section 2.4.4 and Section 3.2).

Finally the matrix C^{seed} is searched for the MFE :

$$MFE = \min_{i,j,K} \left\{ \quad C^{seed}_{dangle}(i, j, K) \quad \right\}.$$

The MFE structure can be computed by a traceback on the matrices.
The time complexity of the algorithm is $O(nmV^2)$. This is because we have to fill $n * m$ arrays $C(i, j)$. For each array $C(i, j)$ we need to get V minimum values and ensure that those values correspond to different ends, which requires time $O(V^2)$ (more details are provided in the next section). The space complexity of this algorithm is $O(nmV)$. However results in Chapter 5 show that V does not need to be large even for sequences that are 250 nt. long. The default value for V is 5. The algorithm offers great flexibility concerning the tradeoff between time and accuracy; setting V to a small number will reduce the execution time and the accuracy, while setting V to a larger number will increase the required time and the resulting accuracy.

4.3 Implementation

4.3.1 Filling the matrices

We use two 3-D matrices C and C^{seed}. The three dimensions of the matrices have lengths n,m and V, respectively. V is either defined by the user or set to the default value of 5. An entry in each of the matrices($C(i, j, K)$ or $C^{seed}(i, j, K)$) will not only hold an energy value, but will also store the right end base pair (k, l) of the corresponding interaction. We use a *struct* type called *interaction_parameters* for this purpose. C and C^{seed} then become 3-D matrices of *interaction_parameters* structs. Each struct holds three values : *energy*, k and l. We use the dot syntax to denote a specific value in the struct. For example $C(i, j, K)$.*energy* denotes the energy value of the interaction with left end (i, j) and the K^{th} best right end. By this, the matrices *che* and *che*^{seed} can be

abandoned. Therefore $che_1(i, j, K)$ and $che_2(i, j, K)$ can now be denoted by $C(i, j, K).k$ and $C(i, j, K).l$, respectively. The same applies to C^{seed}.
Algorithm 9 shows the order in which the matrix C is filled.

Algorithm 9 The order of filling the matrix C

```
for $i = n$ to 1 do
  for $j = m$ to 1 do
    Fill the array $C[i][j]$ using the recursion 4.1
  end for
end for
```

Algorithm 10 shows in detail how an array $C(i, j)$ is filled with V entries.

The implementation is based on the ADT (abstract data type) *Interaction_Set*. *Interaction_Set* stores structs of type *interaction_parameters*. An instance of the ADT is initialized with a specific capacity (V in our case). If more than V elements are pushed into it, the ADT stores only V-many elements, those that have the lowest energy values of all the pushed elements, and the rest of the elements are ignored. Thus it filters the pushed elements to keep only the V best (by energy) elements. The name of the ADT is *Interaction_Set*, since it is a set in the sense that it does not store *interaction_parameters* structs with the same (k, l). If a struct was pushed and there was another struct already in the ADT with the same pair (k, l), the one with the minimum *energy* is taken and the other is ignored. The implementation of the ADT is discussed in Section 4.3.2.

Having the *Interaction_Set* ADT, the array $C(i, j)$ is filled by simply pushing into the *Interaction_Set* all the structs corresponding to the possible extensions of $C(p, q, R)$ (Case (1)). Then we push the struct corresponding to Case (2) when (i, j) is the right most (first) base pair of a new interaction. Then, the *Interaction_Set* holds the V elements with the minimum *energy* values. Afterwards we directly copy the elements from the *Interaction_Set* to the array $C(i, j)$. Similarly we fill the array C^{seed}.

4.3.2 Implementation of *Interaction_Set* ADT

Interaction_Set is an ADT implemented only for the use of the improved heuristic algorithm. It uses an array of structs of type *interaction_parameters* to store the elements. The ADT is given a capacity (denoted by N) when instantiated, which then creates an array of size N. The main functionality of the ADT is that many elements can be pushed into it, but it stores only the N best elements and ignores the rest. The N best elements are the N structs with the lowest energy values. Additionally, the ADT ensures that the stored N structs have distinct right ends (k, l). The interface of the ADT allows two operations : (1) *push* : pushes an element to the ADT, which can be either stored or ignored (2) *read* : given an index of an element, the element with this index in the array is returned, e.g., $Interaction\_Set\_Instance1[K]$ gives the K^{th} element in the array.

Algorithm 10 Filling the array $C(i,j)$

```
if $CannotPair(i,j)$ then
    for $K = 1$ to $V$ do
        $C[i][j][K].energy = \infty$
    end for
else
    Declare data structure $list$ of type $Interaction\_Set$ with size $V$
    Declare struct E of type $interaction\_parameters$
    //Case 1
    for $p = i+1$ to $min(i + s^{loop} + 1, n)$ do
        for $q = j+1$ to $min(j + s^{loop} + 1, m)$ do
            for $R = 1$ to $V$ do
                $E.energy$=$E^{loop}(i,j,p,q) + C[p][q][R].energy$
                                $-ED_1(p, C[p][q][R].k) - ED_2(q, C[p][q][R].l)$
                                $+ED_1(i, C[p][q][R].k) + ED_2(j, C[p][q][R].l)$

                $E.k = C[p][q][R].k$
                $E.l = C[p][q][R].l$

                $list.push(E)$
            end for
        end for
    end for


    // Case 2
    $E.energy$= $ED_1(i,i) + ED_2(j,j)$
    $E.k = i$
    $E.l = j$
    $list.push(E)$

    for $K = 1$ to $V$ do
        $C[i][j][K] = list[K]$
    end for
end if
```

Algorithm 11 Internal implementation of *Interaction_Set* ADT

```
define class Interaction_Set
{
    Declare interaction_parameters Array arr
    Declare integer N
    Declare integer numE
    Declare integer maxEnergy
    function instantiate(integer capacity)
        N = capacity
        Create array arr with size = N
        numE = 0
        max = $-\infty$
    end function

    function push(interaction_parameters e)
        //details in Algorithm 12
    end function
}
```

When an element is pushed, this element should be either stored or ignored. The decision depends on the following :

- if the array is full,
- if the pushed element has an energy less than the maximum energy already stored,
- if an element already stored in the array has the same right end as the pushed element.

Thus to be able to decide, the ADT should keep track of the number of elements stored so far, the maximum energy of the stored elements and check for each push operation if the right end of the pushed element already exists in the array.

Algorithm 11 shows the variables of the ADT :

- *arr* is the array where elements will be stored.
- N is the capacity or the maximum number of elements the ADT should hold.
- *numE* is the number of elements stored in the ADT so far.
- *maxEnergy* is the maximum energy value of all the stored structs.

Algorithm 12 function push(*interaction_parameters* E)

```
1:  if $numE = 0$ then
2:      $arr[1] = E$
3:      $max = E.energy$
4:      $numE{+}{+}$
5:      return
6:  end if
7:
8:  Declare boolean newRE = checkNewRightEnd($E.k$,$E.l$) //$O(N)$
9:
10: if (newRE= $false$) then
11:     Declare $interaction\_parameters$ struct $E_{sameRE}$
12:     $E_{sameRE}$=getElementByRE($E.k$,$E.l$) //$O(N)$
13:     if ($E.energy < E_{sameRE}$) then
14:         replace ($E_{sameRE}$) by ($E$)
15:         $max$=getMaxEnergy()                    //$O(N)$
16:     end if
17: else if (newRE= $true$) and ($numE < N$) then
18:     $arr[numE + 1] = E$
19:     $numE{+}{+}$
20:     if ($E.energy > max$) then
21:         $max = E.energy$
22:     end if
23: else if (newRE= $true$) and ($numE = N$) and ($E.energy < max$) then
24:     replace (array element with $max$ energy) by ($E$)
25:     $max$=getMaxEnergy() //$O(N)$
26: else
27:     Ignore $E$
28: end if
```

Algorithm 12 shows in detail how the function *push* works, and how it keeps in the array only the best N elements of all the pushed elements.

In Line 1 → 6, it is checked if the the current push operation is the first one, and thus we store the pushed struct E regardless of its values. If the push operation is not the first, then we check if the pushed element has a new right end, (i.e. there is no element in the array with the same right end). This is done by the function checkNewRightEnd given (k, l) of E. This function requires time $O(N)$ because it simply tests every element in the array. The result of the function can either be $true$ or $false$ and this result is saved in a boolean variable $newRE$ (new Right End). After that it can be decided whether E will be stored or ignored.

In only three distinct cases E will be stored, otherwise it will be ignored. Case 1 (Line 10) is observed when E has a right end that already exist in the array. Then the two elements of the same right end (E and the element stored in the array denoted by E_{sameRE}) should be compared by their energy. If E has less energy then it replaces E_{sameRE}.

Looking for E_{sameRE} in the array require $O(N)$ by the function getElementByRE. This function could be merged with the function checkNewRightEnd and called only once in the beginning, but it is presented this way for simplicity. In case 2 (Line 17), the array is not full and the pushed element has a new right end (k, l). Then we directly store the element, and check if the maximum energy max should be updated. Case 3 (Line 23) is when the array is already full, E has a better (less) energy than the worst element in the array and it also has a new right end. In this case E should not be added to the array, but rather replaces another element to avoid exceeding the maximum capacity N. E replaces the element with the worst energy in the array. Then to maintain a correct value for max, the array is searched for the maximum energy value in time $O(N)$ by the function getMaxEnergy(). E is ignored if does not fall in one of the cases mentioned above. A flow chart is provided in Figure 4.4 to clarify the three cases in a different (but equivalent) way.

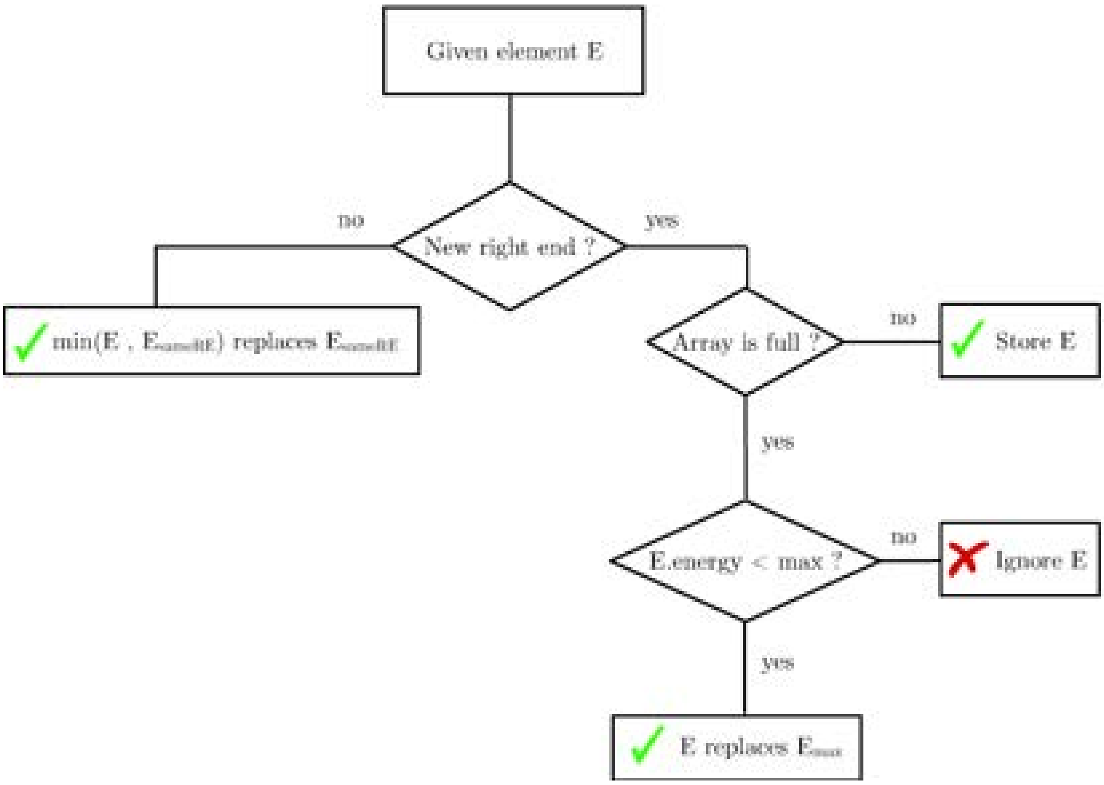


Figure 4.4: The flow chart illustrates the different cases that can occur when a new element is pushed in the ADT.

It can be seen that a *push* operation has a worst case time complexity $O(N)$. Algorithm 10 in the previous section shows that filling an array $C(i, j)$ does a number of push operations proportional to V, and since each of them need $O(V)$ time, therefore filling an array $C(i, j)$ requires $O(V^2)$. Consequently, filling all $n * m$ arrays requires $O(nmV^2)$ time.

Chapter 5

Results and discussion

5.1 Performance of the improved heuristic algorithm

To test and evaluate the improved heuristic algorithm, a small experiment was performed with a pair of artificial RNAs. The algorithm was executed with different values for V which controls the number of different interaction starts that are considered. The energy values for each execution were compared with the energy resulting from the non-heuristic algorithm. The following are the artificial RNAs that were used :

RNA 1 : UCGAGCAGAGAGAGAGAGAGCAGCGGAUCAUUACGACUUAUCGAUGACGGCUACGAUCG
RNA 2 : UACGGACGACGCAGCAUCAUCGAUCG

Algorithm	**V**	**Energy(kcal/mol)**
Improved heuristic version	1	-2.741
	2	-3.921
	3	-5.644
Non-heuristic version		-5.644

Table 5.1: Energy outputs for the heuristic *IntaRNA* algorithm with varying number of interaction starts (V) and the non-heuristic *IntaRNA* algorithm.

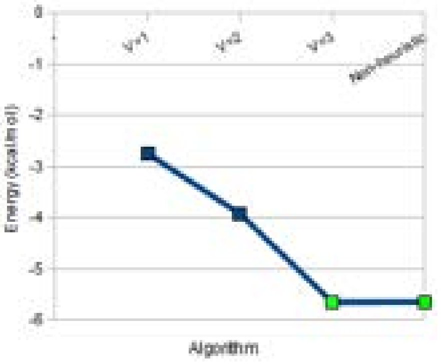


Figure 5.1: A graph for energy outputs according to Table 5.1

The experiment illustrates the relation between V and the resulting energy for the two tested short RNAs. It is clear that increasing the number of considered interaction right ends also increases the number of structures considered, which in turn allows for more stable structures to be covered.

5.2 Performance on prediction of bacterial RNA-RNA interactions

Next, we evaluated and compared the different algorithms of *IntaRNA*. The evaluation included 36 RNA-RNA interactions for which experimentally validated structures are available (see Table 5.1). The test set included 25 interactions for RNAs from *Escherichia coli* and 11 from *Salmonella* (both are bacterial organisms). The evaluation included three different algorithms with various parameter settings. The predicted interactions were evaluated in terms of *Sensitivity*, *PPV* and *F-measure*, where *Sensitivity*=$\frac{\text{number of correctly predicted base pairs}}{\text{number of true base pairs}}$, $PPV=\frac{\text{number of correctly predicted base pairs}}{\text{number of predicted base pairs}}$ and *F-measure*=$\frac{2*Sensitivity*PPV}{Sensitivity+PPV}$.

The following algorithm versions were used : heuristic version, improved heuristic version and the new non-heuristic version. There was no restriction on the interaction length and the maximum loop size was set to 16 (except for one experiment). The improved heuristic version was used with parameter V equal to 1, 2, 3, 4, 5, 6, 8 and 12. Additionally we applied the heuristic version with $V = 3$ and a maximal loop size of 9. All experiments were performed twice, once with a seed region of at least 7 paired nucleotides ($P = 7$), and once with at least 8 paired nucleotides allowing at most one unpaired base in both sequences ($P = 8, b^{max} = 1$). Two additional parameters were used for all executions which are related to the computation of ED values of the target RNA using *RNAplfold*. First, the window size for the sliding window approach - when the target RNA is locally folding - was set to 140 (w=140) instead of the default value of 70. Second, the maximum distance between two paired bases in the target RNA was set to 70 (l=70).

Table 5.2 shows the *F-measures* for $P = 7$ and Table 5.3 shows the *F-measures* when $P = 8$ and $b^{max} = 1$. The length of target RNAs is 250, and the length of the sRNAs varies from 72 to 227. The average runtime on a CPU is shown for each setting.

5.3 Results

An interaction in the table will be refered to by its row number, an algorithm version by its column letter and a specific table entry by both row number and column letter (e.g. A1 refers to the F-measure of the interaction OmrA-cirA predicted by the non-heuristic version).

$P = 7$								
#	Organism+Ref	sRNA—target	**Non-heuristic**	**Improved heuristic**				**Heuristic**
				V=3		V=2	V=1	
				s^{loop}=9		s^{loop}=16		
			A	B	C	D	E	F
1	E.coli[13]	OmrA-cirA	0	0	0	0	0	0
2	E.coli[13]	OmrB-cirA	0.61	0.61	0.61	0.61	0.61	0.61
3	E.coli[3]	OxyS-fhlA	0.593	0.593	0.593	0.593	0.593	0.593
4	E.coli[30]	RyhB-fur	0	0	0	0	0	0
5	E.coli	Spot42-galK	0.523	0.506	0.506	0.523	0.524	0.523
6	E.coli[29]	GlmZ-glmS	1	1	1	1	1	1
7	E.coli[17]	DsrA-hns	0.69	0.69	0.69	0.69	1	0.828
8	E.coli[11]	CyaR-luxS	0.889	0.889	0.889	0.889	0.889	0.889
9	E.coli[11]	CyaR-nadE	-	-	-	-	-	-
10	E.coli[28]	MicA-ompA	0.897	0.897	0.897	0.897	0.897	0.897
11	E.coli	MicC-ompC	0.667	0.478	0.667	0.667	0.667	0.667
12	E.coli[26]	MicF-ompF	0.96	0.96	0.96	0.96	0.96	0.96
13	E.coli[13]	OmrA-ompR	0.75	0.75	0.75	0.75	0.75	0.75
14	E.coli[13]	OmrB-ompR	0.783	0.783	0.783	0.783	0.783	0.783
15	E.coli[13]	OmrA-ompT	0.632	0.632	0.632	0.632	0.632	0.632
16	E.coli[13]	OmrB-ompT	0	0	0	0	0	0
17	E.coli[11]	CyaR-ompX	0.529	0.529	0.529	0.529	0.529	0.529
18	E.coli[16]	SgrS-ptsG	0.85	0.85	0.85	0.85	0.85	0.85
19	E.coli[18]	DsrA-rpoS	0.792	0.792	0.792	0.792	0.792	0.792
20	E.coli[19]	RprA-rpoS	0.75	0.75	0.75	0.75	0.75	0.75
21	E.coli	RyhB-sdhD	0.871	0.871	0.871	0.867	0.867	0.741
22	E.coli	RyhB-shiA	0.491	0.491	0.491	0.491	0.491	0.491
23	E.coli[12]	RyhB-sodB	0.621	0.621	0.621	0.621	0.621	0.621
24	E.coli[24]	GcvB-sstT	0	0	0	0	0	0
25	E.coli[11]	CyaR-yqaE	0.643	0.643	0.643	0.643	0.643	0.643
26	Salmonella[27]	GcvB-argT	0.941	0.882	0.882	0.889	0.914	0.941
27	Salmonella[27]	GcvB-dppA	0.68	0.68	0.68	0.68	0.68	0.68
28	Salmonella[27]	GcvB-gltI	0	0	0	0	0.706	0
29	Salmonella[8]	MicA-lamB	0.902	0.902	0.902	0.902	0.902	0.902
30	Salmonella[27]	GcvB-livJ	0.955	1	1	1	0	0.955
31	Salmonella[27]	GcvB-livK	0.722	0.722	0.722	0.722	0.722	0.722
32	Salmonella[23]	MicC-nmpC	0.957	0.957	0.957	0.957	0.957	0.957
33	Salmonella	RybB-ompN	0.857	0.857	0.857	0.857	0.857	0.857
34	Salmonella[22]	CyaR-ompX	0.478	0.478	0.478	0.478	0.478	0.478
35	Salmonella[27]	GcvB-oppA	0.978	0.955	0.955	0.955	0.955	0.978
36	Salmonella[27]	GcvB-STM4351	0.588	0.529	0.529	0	0	0
Average F-Measure			**0.6278**	**0.6194**	**0.6246**	**0.6105**	**0.6116**	**0.6116**
Average runtime			16.5 min	1.15 sec	2.06 sec	1.63 sec	0.9 sec	2.17 sec
Average F-Measure without row 9			**0.6636**	**0.6548**	**0.6603**	**0.6454**	**0.6466**	**0.6466**
Average interaction energy			**-13.85**	**-13.47**	**-13.83**	**-13.76**	**-13.61**	**-13.67**

Table 5.2: F-measures for $P = 7$

When we increase the parameter V for the improved heuristic version, the predicted MFE structures have lower energies (more stable) and tend to give better F-measures. For the experiment with $P = 7$, the improved heuristic version with $V \geq 5$ has the same energy results as the non-heuristic version because for $V = 5$ the improved heuristic algorithm could already get an MFE structure. The same applies to the experiment with $P = 8$ and $b^{max} = 1$ but with $V \geq 3$. This supports a previous argument in Chapter 4 that V need not be large to get MFE structures. However, the F-measures for the improved heuristic version with $V = 5$ are the same as with $V = 4$ or $V = 3$. Hence, we present here results for $V = 1, 2$ and 3 only.

It should be noted that the difference in F-measure in some examples for the heuristic version and the improved heuristic version with $V = 1$ results from different computation of dangling end energies. However they have an equal average F-measure.

Discussion on results of Table 5.2

No interaction between Cyar and nadE (row 9) could be predicted since the interaction does not satisfy the seed restriction of 7 consecutive base pairings. Therefore we calculated an average F-measure without this particular interaction to allow a fair comparison between Table 5.2 and Table 5.3.

The difference in energy between the improved heuristic with $V = 3$ and the non-heuristic occurs only in interactions with *F-measure* $= 0$. For the rows with non-zero *F-measures*, only the two experimental settings in Row A and C yielded MFE structures for all interactions. However A has a better F-measure because interactions predicted from both settings were not exactly the same. When the predictions were further analyzed, it was found that for all the interactions in which A is better than C (Rows 5, 26, 35 and 36), the predictions of C were in the exact same regions in both sRNA and target as the predictions of A; only the base pairs were different. This means that the way the non-heuristic version forms the interaction (in both directions) is better than extending an interaction only to the left.

The non-heuristic version always gets the MFE structure and is, in turn, time consuming. However, although Setting C is better than F, the difference is very small, which does not leave much space for improvements over the previous heuristic version. In some cases, thermodynamically less stable interactions score a better F-measure than more stable ones, this can be seen in (F5,C5), (F7,C7) and (A30,C30).

For the predicted interactions in Row 28 the F-measure is always zero except for setting E. When those interactions were examined, we found out that the interacting region of the sRNA was the same as in the validated experiment. However the interacting region of the target was wrong, and therefore no base pairs matched the true ones. But this means that looking for *non-overlapping* suboptimal results would never lead to better F-measure, simply because the true interaction overlaps the optimally predicted one in the sRNA. We conclude that *overlapping* suboptimal results could be useful in some cases. Setting E gave an F-measure of 0.706 because the MFE structure was not found, but instead a suboptimal structure was found that matched the validated interaction in most of the base pairs.

The purpose of Setting B was to simulate stopping interactions at energy barriers constituted by long loops, which are usually destabilizing structures. In C11, C17 and C24, the predicted structures had loops longer than 9. In Setting B the prediction gave different structures with less stability. However, this only affected the F-measure of C11. In two of the three cases (17 and 24) the structures predicted using $s^{loop} = 9$ had the long loops split into two parts (by one or two base pairs) and the rest of the interaction was almost the same.

In Row 30, Settings B, C and D perfectly predicted the interaction. Setting A and F got the true interaction region in both sequences, however there was one mismatching base pair. Setting E had the interacting regions shifted by 7 nucleotides in the sRNA and 4 nucleotides in the target, therefore all base pairs mismatched the true interaction.

All the interactions have only one hybridization region, except for OxyS-fhlA that has two regions. *IntaRNA* could predict one of the two regions with good accuracy. If we neglect the unpredicted region, then the Sensitivity for the predicted region equals 0.89, the PPV equals 0.73 and the F-measure equals 0.8.

$P = 8$, $b^{max} = 1$								
#	Organism	sRNA—target	**Non-heuristic**	**Improved heuristic**				**Heuristic**
				V=3		V=2	V=1	
				s^{loop}=9		s^{loop}=16		
			A	B	C	D	E	F
1	E.coli	OmrA-cirA	0	0	0	0	0	0
2	E.coli	OmrB-cirA	0.61	0.61	0.61	0.61	0.61	0.61
3	E.coli	OxyS-fhlA	0.593	0.593	0.593	0.593	0.593	0.593
4	E.coli	RyhB-fur	0	0	0	0	0	0
5	E.coli	Spot42-galK	0.523	0.506	0.506	0.523	0.524	0.523
6	E.coli	GlmZ-glmS	1	1	1	1	1	1
7	E.coli	DsrA-hns	0.828	0.69	0.69	0.69	1	0.828
8	E.coli	CyaR-luxS	0.889	0.889	0.889	0.889	0.889	0.889
9	E.coli	CyaR-nadE	0.952	0.952	0.952	0.741	0.741	0.952
10	E.coli	MicA-ompA	0.897	0.897	0.897	0.897	0.897	0.897
11	E.coli	MicC-ompC	0.667	0.478	0.667	0.667	0.667	0.667
12	E.coli	MicF-ompF	0.96	0.96	0.96	0.96	0.96	0.96
13	E.coli	OmrA-ompR	0.75	0.75	0.75	0.75	0.75	0.75
14	E.coli	OmrB-ompR	0.783	0.783	0.783	0.783	0.783	0.783
15	E.coli	OmrA-ompT	0.632	0.632	0.632	0.632	0.632	0.632
16	E.coli	OmrB-ompT	0	0	0	0	0	0
17	E.coli	CyaR-ompX	0.529	0.529	0.529	0.529	0.529	0.529
18	E.coli	SgrS-ptsG	0.85	0.85	0.85	0.85	0.85	0.85
19	E.coli	DsrA-rpoS	0.792	0.792	0.792	0.792	0.792	0.792
20	E.coli	RprA-rpoS	0.4	0.4	0.4	0.4	0.4	0.4
21	E.coli	RyhB-sdhD	0.871	0.871	0.871	0.867	0.867	0.741
22	E.coli	RyhB-shiA	0.491	0.491	0.491	0.491	0.491	0.491
23	E.coli	RyhB-sodB	0.621	0.621	0.621	0.621	0.621	0.621
24	E.coli	GcvB-sstT	0	0	0	0	0	0
25	E.coli	CyaR-yqaE	0.643	0.643	0.643	0.643	0.643	0.643
26	Salmonella	GcvB-argT	0.882	0.882	0.882	0.889	0.857	0.882
27	Salmonella	GcvB-dppA	0.64	0.68	0.68	0.68	0.68	0.68
28	Salmonella	GcvB-gltI	0	0	0	0	0.706	0
29	Salmonella	MicA-lamB	0.902	0.902	0.902	0.902	0.902	0.902
30	Salmonella	GcvB-livJ	0.955	0.955	0.955	0.955	0.5	0.955
31	Salmonella	GcvB-livK	0.722	0.722	0.722	0.722	0.722	0.722
32	Salmonella	MicC-nmpC	0.957	0.957	0.957	0.957	0.957	0.957
33	Salmonella	RybB-ompN	0.857	0.857	0.857	0.857	0.857	0.857
34	Salmonella	CyaR-ompX	0.478	0.478	0.478	0.478	0.478	0.478
35	Salmonella	GcvB-oppA	0.978	0.955	0.955	0.955	0.955	0.978
36	Salmonella	GcvB-STM4351	0.588	0.529	0.529	0.529	0.529	0.529
Average F-Measure			**0.6456**	**0.6348**	**0.6401**	**0.6348**	**0.6495**	**0.6414**
Average runtime			18.14 min	1.37 sec	2.29 sec	1.84 sec	1.11 sec	2.38 sec
Average F-Measure without row 9			**0.6552**	**0.6439**	**0.6495**	**0.6499**	**0.6654**	**0.6509**
Average interaction energy			**-14**	**-13.58**	**-14**	**-13.94**	**-13.78**	**-13.84**

Table 5.3: F-measures for $P = 8$, $b^{max} = 1$

Discussion on results of Table 5.3

In Table 5.3 the interaction CyaR-nadE in Row 9 could be predicted with very good accuracy due to the weaker seed conditions.

Comparing the averages of both Table 5.2 and Table 5.3 without row 9, we find that results for $P = 7$ are better for settings A, B and C, while for settings D, E and F, $P = 8$, $b^{max} = 1$ leads to better results.

5.4 Time and space requirements

The improved heuristic version could get a very close result to the non-heuristic version with drastically reduced time. To conduct the 36 experiments sequentially, the non-heuristic version took almost 10 hours, while the improved heuristic version with $V = 3$ took only 74 seconds (500 times faster). The previous heuristic version took 78 seconds.

The memory consumption of the non-heuristic version ranges from 390 MB to 3.1 GB with an average of 1.14 GB, while for the improved heuristic version with $V = 3$ the maximum consumption was 23 MB for the largest input SgrS-ptsG which has $n = 227$ and $m = 250$.

Figure 5.2 shows two graphs for comparison between the non-heuristic and the improved heuristic versions.

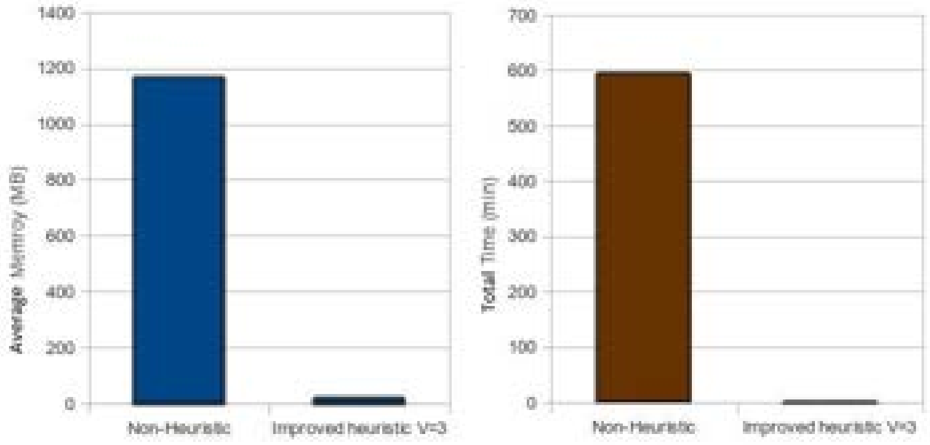


Figure 5.2: Comparison between memory and time requirements of non-heuristic and improved heuristic *IntaRNA* algorithm. The first graph is for the average memory used per prediction, the second is for the total time taken to perform the 36 predictions.

5.5 Future work on the non-heuristic algorithm

The motivation for implementing this algorithm was to detect intermediate points in interaction formation at which adding further base pairs would require relatively high amounts of energy due to the unfolding energy of the interaction sites. The algorithm is ready for adding this feature, however a threshold should be calculated to be able to determine whether the energy needed to add a base pair is too high, in this case, the interaction should be stopped just before adding this base pair. This threshold value is not available at the moment. Therefore this feature could not be tested. In Figure 5.3, the energies of an example interaction were plotted against interaction length, each point in the figure represents one added base pair.

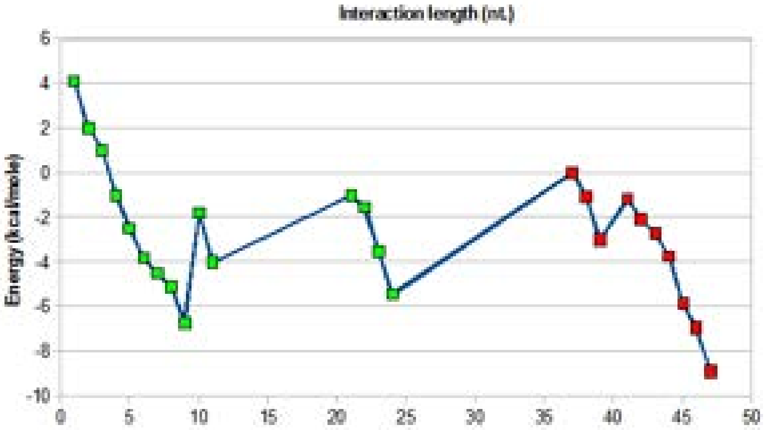


Figure 5.3: Energy plot of intermediate states in interaction formation. Each point on the figure represents one added base pair to the interaction.

After adding the green base pairs, high amount of energy is required to add the next base pair (due to a long loop in the interaction). Therefore we could stop the interaction after forming the green base pairs if the energy difference between the last green and the first red point shown was greater than the threshold.

The threshold can be calculated statistically using data sets of interactions, then determine the interactions where the predicted structures had longer interaction regions than in the true interaction, and finally study the energy differences in the point at which the true interaction formation stops.

When the threshold is known, the feature can be implemented by modifying the recursion (Equation 3.1). For filling the entry $C(i,j,k,l)$, the minimum energy is taken over all possible extensions to the left and right. Suppose it was found that it is best to extend $C(p,q,k,l)$ to the left, then the modification is to get the energy difference between the interaction in $C(p,q,k,l)$ before and after the extension. If this energy difference exceeds the threshold, then $C(i,j,k,l)$ is set to ∞.

Another possibility to modify the recursion is not to take the minimum over all possible extensions, but instead, take the minimum over only those extensions that have an energy difference (energy before and after extending) less than or equal to the threshold.

As already mentioned in the results of Setting B, applying the threshold may only result in splitting the long loop by a few base pairs as in Figure 5.4. This shows that it might be necessary to apply energy threshold to intervals and not just 2 consecutive base pairs.

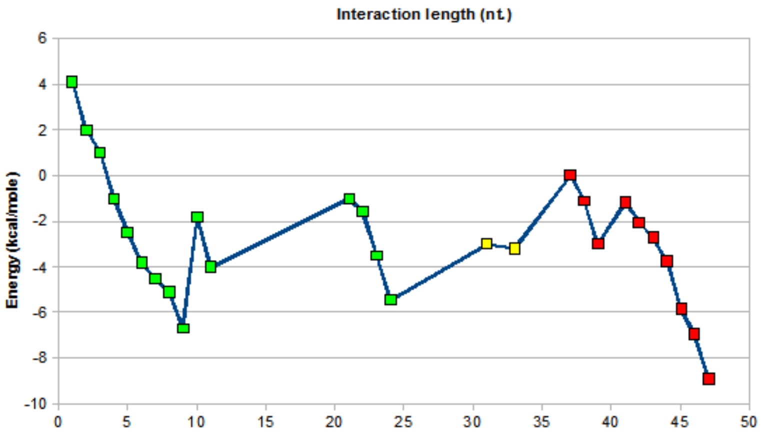


Figure 5.4: The energy plot shows an example interaction after applying energy threshold. The result of applying the energy threshold was splitting a long loop by 2 base pairs.

5.6 Conclusion

We presented an improved heuristic and a non-heuristic algorithm for *IntaRNA*. The evaluation showed that the existing heuristic version has already a high accuracy, which could be only marginally improved. However the developed improved heuristic algorithm could get structures with the same energy as the non-heuristic algorithm in drastically less time.

In a number of cases, energetically less stable interactions scored better - in terms of accuracy - than more stable ones, i.e., the correlation between energy stability and prediction accuracy is not strong.

Future research should be done on detecting energy barriers to improve the accuracy of the current prediction model. Furthermore, the prediction model should be extended by additional features.

Bibliography

[1] C. Alkan, E. Karakoc, J.H. Nadeau, S.C. Şahinalp, and K. Zhang. RNA-RNA interaction prediction and antisense RNA target search. In *Research in Computational Molecular Biology*, pages 152–171. Springer, 2005.

[2] M. Andronescu, R. Aguirre-Hernandez, A. Condon, and H.H. Hoos. RNAsoft: a suite of RNA secondary structure prediction and design software tools. *Nucleic Acids Research*, 31(13):3416, 2003.

[3] L. Argaman and S. Altuvia. fhlA repression by OxyS RNA: kissing complex formation at two sites results in a stable antisense-target RNA complex1. *Journal of Molecular Biology*, 300(5):1101–1112, 2000.

[4] R. Backofen and W.R. Hess. Computational prediction of sRNAs and their targets in bacteria. *RNA Biol*, 7:1–10, 2010.

[5] S.H. Bernhart, I.L. Hofacker, and P.F. Stadler. Local RNA base pairing probabilities in large sequences. *Bioinformatics*, 2005.

[6] S.H. Bernhart, H. Tafer, U. Mückstein, C. Flamm, P.F. Stadler, and I.L. Hofacker. Partition function and base pairing probabilities of RNA heterodimers. *Algorithms for Molecular Biology*, 1(1):3, 2006.

[7] A.F. Bompfünewerer, R. Backofen, S.H. Bernhart, J. Hertel, I.L. Hofacker, P.F. Stadler, and S. Will. Variations on RNA folding and alignment: lessons from Benasque. *Journal of Mathematical Biology*, 56(1):129–144, 2008.

[8] L. Bossi and N. Figueroa-Bossi. A small RNA downregulates LamB maltoporin in Salmonella. *Molecular microbiology*, 65(3):799–810, 2007.

[9] A. Busch, A.S. Richter, and R. Backofen. IntaRNA: efficient prediction of bacterial sRNA targets incorporating target site accessibility and seed regions. *Bioinformatics*, 24(24):2849, 2008.

[10] H. Chitsaz, R. Salari, S.C. Sahinalp, and R. Backofen. A partition function algorithm for interacting nucleic acid strands. *Bioinformatics*, 25(12):i365, 2009.

[11] N. De Lay and S. Gottesman. The Crp-activated small noncoding regulatory RNA CyaR (RyeE) links nutritional status to group behavior. *Journal of bacteriology*, 191(2):461, 2009.

[12] T.A. Geissmann and D. Touati. Hfq, a new chaperoning role: binding to messenger RNA determines access for small RNA regulator. *The EMBO Journal*, 23(2):396–405, 2004.

[13] M. Guillier and S. Gottesman. The 5'end of two redundant sRNAs is involved in the regulation of multiple targets, including their own regulator. *Nucleic Acids Research*, 2008.

[14] I.L. Hofacker. Vienna RNA secondary structure server. *Nucleic acids research*, 31(13):3429, 2003.

[15] F.W.D. Huang, J. Qin, C.M. Reidys, and P.F. Stadler. Partition function and base pairing probabilities for RNA-RNA interaction prediction. *Bioinformatics*, 25(20):2646, 2009.

[16] H. Kawamoto, Y. Koide, T. Morita, and H. Aiba. Base-pairing requirement for RNA silencing by a bacterial small RNA and acceleration of duplex formation by Hfq. *Molecular microbiology*, 61(4):1013–1022, 2006.

[17] R.A. Lease, M.E. Cusick, and M. Belfort. Riboregulation in Escherichia coli: DsrA RNA acts by RNA: RNA interactions at multiple loci. *Proceedings of the National Academy of Sciences of the United States of America*, 95(21):12456, 1998.

[18] N. Majdalani, C. Cunning, D. Sledjeski, T. Elliott, and S. Gottesman. DsrA RNA regulates translation of RpoS message by an anti-antisense mechanism, independent of its action as an antisilencer of transcription. *Proceedings of the National Academy of Sciences of the United States of America*, 95(21):12462, 1998.

[19] N. Majdalani, D. Hernandez, and S. Gottesman. Regulation and mode of action of the second small RNA activator of RpoS translation, RprA. *Molecular microbiology*, 46(3):813–826, 2002.

[20] J.S. McCaskill. The equilibrium partition function and base pair binding probabilities for RNA secondary structure. *Biopolymers*, 29(6-7):1105–1119, 1990.

[21] U. Muckstein, H. Tafer, J. Hackermuller, S.H. Bernhart, P.F. Stadler, and I.L. Hofacker. Thermodynamics of RNA-RNA binding. *Bioinformatics*, 22(10):1177, 2006.

[22] K. Papenfort, V. Pfeiffer, S. Lucchini, A. Sonawane, J.C.D. Hinton, and J. Vogel. Systematic deletion of Salmonella small RNA genes identifies CyaR, a conserved CRP-dependent riboregulator of OmpX synthesis. *Molecular microbiology*, 68(4):890–906, 2008.

[23] V. Pfeiffer, K. Papenfort, S. Lucchini, J.C.D. Hinton, and J. Vogel. Coding sequence targeting by MicC RNA reveals bacterial mRNA silencing downstream of translational initiation. *Nature Structural & Molecular Biology*, 16(8):840–846, 2009.

[24] S.C. Pulvermacher, L.T. Stauffer, and G.V. Stauffer. Role of the Escherichia coli Hfq protein in GcvB regulation of oppA and dppA mRNAs. *Microbiology*, 155(1):115, 2009.

[25] M. Rehmsmeier, P. Steffen, M. Höchsmann, and R. Giegerich. Fast and effective prediction of microRNA/target duplexes. *Rna*, 10(10):1507, 2004.

[26] M. Schmidt, P. Zheng, and N. Delihas. Secondary structures of Escherichia coli antisense micF RNA, the 5'-end of the target ompF mRNA, and the RNA/RNA duplex. *Biochemistry*, 34(11):3621–3631, 1995.

[27] C.M. Sharma, F. Darfeuille, T.H. Plantinga, and J. Vogel. A small RNA regulates multiple ABC transporter mRNAs by targeting C/A-rich elements inside and upstream of ribosome-binding sites. *Genes & development*, 21(21):2804, 2007.

[28] K.I. Udekwu, F. Darfeuille, J. Vogel, J. Reimegård, E. Holmqvist, and E.G.H. Wagner. Hfq-dependent regulation of OmpA synthesis is mediated by an antisense RNA. *Genes & development*, 19(19):2355, 2005.

[29] J.H. Urban and J. Vogel. Two seemingly homologous noncoding RNAs act hierarchically to activate glmS mRNA translation. *PLoS Biol*, 6(3):e64, 2008.

[30] B. Večerek, I. Moll, and U. Bläsi. Control of Fur synthesis by the non-coding RNA RyhB and iron-responsive decoding. *The EMBO journal*, 26(4):965, 2007.