Safe and Complete Prediction of RNA Secondary Structure

Niko Kiirala

Helsinki January 12, 2019

UNIVERSITY OF HELSINKI
Department of Computer Science

HELSINGIN YLIOPISTO — HELSINGFORS UNIVERSITET — UNIVERSITY OF HELSINKI

Tiedekunta — Fakultet — Faculty	Koulutusohjelma — Studieprogram — Study Programme
Faculty of Science	Computer Science

Tekijä — Författare — Author
Niko Kiirala

Työn nimi — Arbetets titel — Title

Safe and Complete Prediction of RNA Secondary Structure

Ohjaajat — Handledare — Supervisors
Leena Salmela and Alexandru Tomescu

Työn laji — Arbetets art — Level	Aika — Datum — Month and year	Sivumäärä — Sidoantal — Number of pages
Master's Thesis	January 12, 2019	54 pages + 2 appendices

Tiivistelmä — Referat — Abstract

Ribonucleic acid, RNA, is an essential type of molecule for all known forms of life. It is a nucleic acid, like DNA. However, where DNA appears as two complementary strands that join and twist into a double helix structure, RNA has only a single strand. This strand can fold upon itself, pairing complementary bases. The resulting set of base pairs is the *RNA secondary structure*, also known as *folding*.

It is typical that a prediction algorithm gives a large number of optimal or near-optimal foldings for an RNA sequence. Only in the simplest cases it is possible to manually go through all of these foldings, and in hard cases it is infeasible to even generate the full set of optimal foldings. In fact, we observe that the number of optimal foldings may be exponential in the sequence length, and that some naturally occurring RNA sequences of 2000–3000 bases in length have well over 10^{100} optimal foldings, under the model of maximizing the number of base pairs.

To help analyze the full set of optimal foldings, we apply the concept of safe and complete algorithms. In the presence of multiple optimal solutions, any partial solution that appears in all optimal solutions is called a *safe* part, and a *safe and complete* algorithm finds all of the safe parts.

We show a trivial safe and complete algorithm that computes safety by going through the full set of optimal foldings. However, this algorithm is only practical for short RNA sequences that do not have too many optimal foldings. In order to analyze the harder RNA sequences, we develop and implement a novel polynomial-time safe and complete algorithm for RNA secondary structure prediction, using the model of maximizing base pairs. Using the dynamic programming approach, this new algorithm can compute how often each base pair and unpaired base appears in the full set of optimal foldings without having to produce the actual foldings.

Our experimental evaluation shows that the safe parts of a folding are more likely to be biologically correct than the non-safe parts. We observe this both by using our implementation of the efficient safe and complete algorithm and by combining an existing predictor program with the trivial algorithm. As this existing predictor uses a modern minimum free energy model for predicting the RNA foldings, tests using this combination show that safety is a useful property, even beyond the simple maximum pairs model in our implementation.

ACM Computing Classification System (CCS):
Applied computing → Life and medical sciences → Computational biology → Molecular structural biology
Theory of computation → Design and analysis of algorithms → Mathematical optimization → Discrete optimization
Theory of Computation → Design and analysis of algorithms → Algorithm design techniques → Dynamic programming

Avainsanat — Nyckelord — Keywords
RNA secondary structure, RNA folding, Safe solution, Safe and complete algorithm

Säilytyspaikka — Förvaringsställe — Where deposited

Muita tietoja — övriga uppgifter — Additional information

Thesis for the Algorithms, Data Analytics and Machine Learning subprogramme

Contents

1 Introduction

Ribonucleic acid, or RNA for short, is an important building piece of life. Notably, protein synthesis depends on multiple specific types of RNA molecules. An RNA molecule is a long strand of phosphates and ribose sugars, with a base attached to each sugar. The sequence of bases affects what shape the RNA molecule assumes and both the sequence and the shape determine what kind of biological function the molecule has.

A DNA molecule famously forms a double helix structure where the two complementary strands of DNA twist around each other. This double helix structure is also known as the secondary structure of DNA. In comparison, an RNA molecule has only a single strand and cannot produce this helix structure. Instead, the secondary structure of RNA is created when the RNA strand folds upon itself and complementary parts of the same strand attach to each other.

Computer predictions of RNA secondary structure typically produce multiple equally or nearly equally good results. This thesis explores how to handle these multiple results in a modeling algorithm and how to identify the parts of a result that are common to all good results.

Predictions of RNA secondary structure are used for various purposes, such as classifying unknown RNA sequences, studying the evolution of RNA molecules and in nucleic acid design. All of these share the common theme that the shape of the RNA molecule is crucial for its function, but the exact sequence of bases may change.

In this thesis, the concept of *safe and complete algorithm* is used to address the problem of multiple good solutions. To expand upon the meaning of the name, "safe" refers to identifying the parts of a solution that appear in all good solutions and "complete" refers to finding all such parts. A safe and complete algorithm for RNA secondary structure prediction would thus find all base pairs and non-paired bases that appear in all optimal solutions.

2 Biological background

In order to explore RNA secondary structure prediction algorithms, we will first take a look into RNA molecules themselves. We start with an overview of how RNA molecules are formed, then discuss how the secondary structure of such molecule is formed and finally take a look into why one might wish to predict the secondary structure.

2.1 Structure of an RNA molecule

A DNA strand is composed of four distinct bases: adenine (`A`), cytosine (`C`), guanine (`G`) and thymine (`T`). RNA is composed of the same bases, except it contains uracil

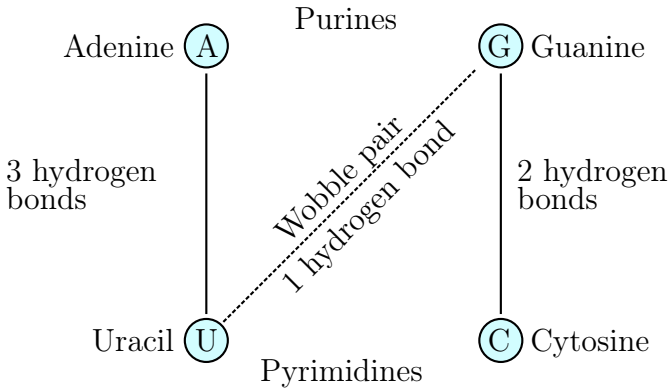


Figure 1: Standard bases of RNA and their pairings

(`U`) instead of thymine. The canonical, or *Watson-Crick*, base pairs of RNA are `A-U` and `C-G`. Other pairings are also possible, but they are energetically unfavourable compared to these. A typical non-canonical pairing in RNA is the *wobble pair* `G-U` which is often considered to be equivalent to standard pairings [Sun10, chapter 11]. The possible pairings are shown in Figure 1.

In addition to the real RNA bases, fictional bases do get used in systems handling RNA data. The IUB/IUPAC nucleic acid codes [CB85] include single-letter codes for each subset of the set of the four standard bases, allowing one to indicate uncertainty or variance of what base is in each position. This thesis makes use of a single fictional base from the IUB/IUPAC list, `N`, which gets used as a non-pairing base.

An RNA strand is built from a backbone chain and bases attached to it. The backbone chain consists of alternating phosphates and ribose sugars. The bases are attached to each ribose of the backbone chain. The combination of single phosphate, ribose and base is called a *nucleotide*. The molecule structure of an RNA nucleotide is shown in Figure 2 [Sun10, chapter 1]. The phosphate group is attached to the $5'$ carbon atom of the ribose sugar and the base to the $1'$ carbon atom. When forming a chain, the `OH` group connected to $3'$ carbon of the ribose and an `OH` group in the phosphate of the next nucleotide combine to form a bond. The polymer formed by connecting multiple nucleotides in such manner is the RNA. By convention, the sequence of bases is read from the end with the phosphate group, called the $5'$ end, towards the end with the ribose, called the $3'$ end.

Adenine and guanine are derivatives of purine and so are known as *purine bases*. Cytosine, thymine and uracil are derivatives of pyrimidine and so are known as *pyrimidine bases*. All typical base pairings pair a purine base with a pyrimidine base. Pyrimidines are the smaller of two types, and combination of two pyrimidine bases is unfavourable because the bonding atoms would be too far from each other. Purines are the larger type of molecule. Two purine bases combined would end up too close to each other and would repel each other.

This thesis focuses on secondary structure, or folding, of RNA molecules. The secondary structure is uniquely defined by listing the base pairs in the molecule.

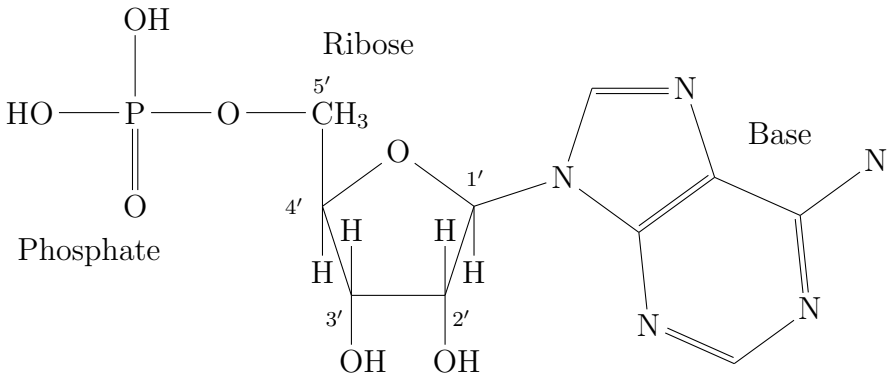


Figure 2: Structure of RNA nucleotide containing adenine base

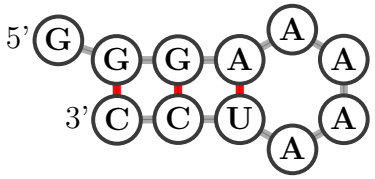


Figure 3: Simple example of RNA folding. Drawn with Forna [KHH15].

RNA also has a *tertiary structure*, which is the three-dimensional shape of the RNA molecule. The topic of tertiary structure is not addressed in this thesis.

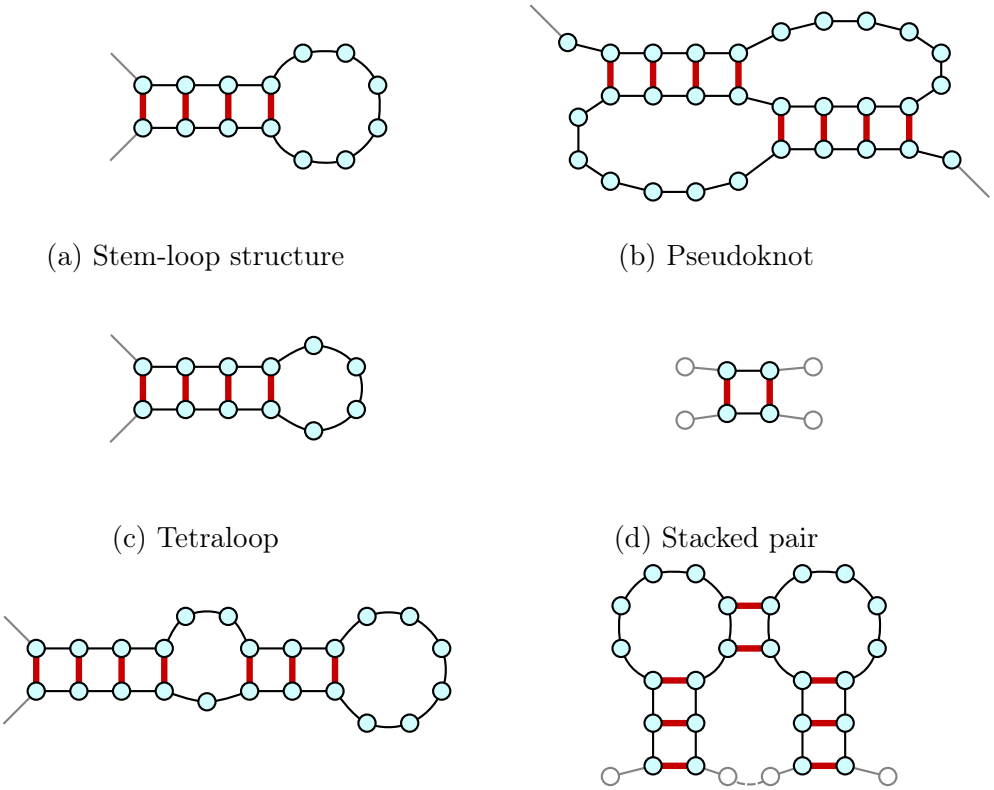
2.2 How RNA folding works

Figure 3 shows a simple example of an RNA folding. This folding is composed of bases `GGGAAAAAUCC` and has three base pairs and a loop of four bases. This folding only contains Watson-Crick base pairs and no wobble pairs.

An RNA strand receives its secondary structure because pairing bases gives the RNA molecule a lower binding energy. Out of all possible secondary structures, the molecule tends to take a form that results in low binding energy.

The *stem-loop* structures and *pseudoknots* are the most common descriptions of RNA structure. A stem-loop structure, as shown in Figure 4a, consists of a stem of base pairs and a loop of unpaired bases in the middle. In a more general sense, we can consider nested stem-loop structures, where the loop may recursively contain further stem-loop structures. A pseudoknot structure, as shown in Figure 4b, connects two parts of a sequence such that they do not form a nested stem-loop structure, but instead connect bases from outside of a loop to the loop. In fact, as this thesis studies only non-pseudoknot structures, any folding can be described in terms of nested stem-loop structures and ranges of unpaired bases.

Tetraloops are stem-loop structures that contain exactly four unpaired bases in the loop, as seen in Figure 4c. Such loops can be highly stable [ALT91]. They can also perform other biological functions. For example a *GNRA tetraloop* can create a sharp turn in the RNA backbone direction by bonding the `G` base with the phosphate



(a) Stem-loop structure

(b) Pseudoknot

(c) Tetraloop

(d) Stacked pair

(e) Internal loop

(f) Kissing hairpins

Figure 4: Examples of motifs in RNA secondary structures

of the A. The bases NRA, where N stands for any base and R stands for G or A, are left facing outwards so they can easily form bonds with external molecules or with other parts of same molecule, forming tertiary structure [JP95].

Beyond stem-loop structures and pseudoknots, there is some variation between different publications in how structures are classified. A stem-loop structure in Figure 4a may be synonymous with a *hairpin loop* [SC06], but this term may also describe a structure that contains only the loop and the single base pair closing the loop [WFHS99]. The stem of a stem-loop structure can be considered to consist of *stacked pairs*, as shown in Figure 4d. When there are free bases inside the stem of a stem-loop structure, on one or both sides of the stem, it can be more natural to consider this as an *internal loop*, as shown in Figure 4e, and not as a nested stem-loop structure. A special case of pseudoknot structures, where the loops of two stem-loop structures connect, is known as *kissing hairpins* shown in Figure 4f.

The predictions made by different algorithms are not completely correct and accurate. The models for binding energy are not perfectly accurate, so the actual binding energy might not match the prediction. Dynamic programming models typically do not attempt to predict pseudoknot structures, as they are more complex to handle. In general case, predicting pseudoknot structures is NP-hard [Aku00]. Also, RNA does not always take the globally optimal shape, but may end up in other shapes due to factors beyond the secondary structure. These factors include the tertiary structure of the resulting molecule and external factors such as concentrations of different ions that may bind to the molecule. For example, RNA from viruses is known to change shape depending on its surroundings [RAR97, GLGW10].

Examining near-optimal solutions may help to find the true secondary structure when the model does not fully match reality. By examining the results from a model, it is possible to determine which parts of the solution are more likely to be correct and which ones are less likely.

2.3 Use cases for RNA structure prediction

The secondary structure of RNA is useful in finding the biological function of a given RNA sequence. When an RNA does not code a protein, but performs another function in the cell, its secondary structure may be what determines its function more than what exact bases it is made of [Sun10, chapter 11]. These structures have been catalogued in order to find similar functional parts between different RNA molecules [THK$^+$04].

A *messenger RNA*, or *mRNA*, is produced when a protein-coding gene is copied from DNA. In mRNA, base triplets, known as *codons*, define the sequence of amino acids that is needed to create the protein [MW97]. The base sequence of mRNA is its most important factor, so this type is not of a particular interest in secondary structure prediction.

This thesis is focused mostly on *transfer RNA*, or *tRNA*. A tRNA molecule is an

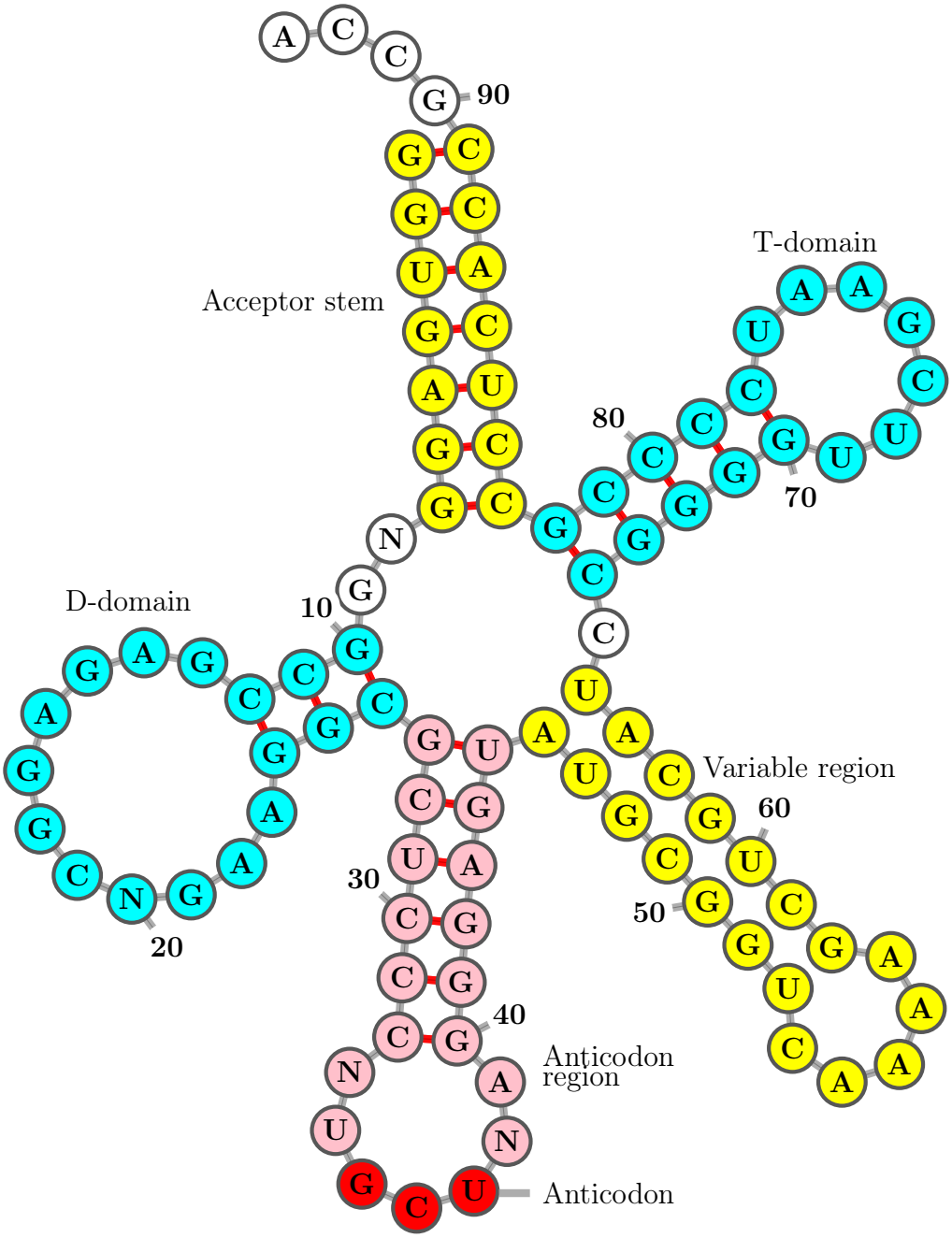


Figure 5: Example of the cloverleaf shape of tRNA (RS1661, from Escherichia coli, from the Sprinzl tRNA database [SSHS96]) Drawn with Forna [KHH15].

essential part in protein synthesis. An example of tRNA and its different regions is shown in Figure 5. The anticodon part of tRNA matches a codon in mRNA. The acceptor stem brings in the matching amino acid that is needed for constructing the protein [MW97]. The secondary structure of tRNA is highly important for its function, so it makes sense to predict it. They are also relatively small molecules at around 80 bases and do not contain pseudoknots. These two properties make tRNA molecules simple and fast to analyze.

Another important RNA type in this thesis is *ribosomal RNA*, shortened as *rRNA*. A ribosome is another crucial part in protein synthesis: this is where an mRNA molecule is attached during protein synthesis and where the tRNA molecules bring the corresponding amino acids. A ribosome consists of multiple rRNA molecules and proteins [MW97]. There are multiple different categories of rRNA molecules, categorized by their size. This thesis examines 5s, 16s and 23s rRNA, which are the types present in prokaryotes, that is, domains bacteria and archaea. The 5s rRNA are around 120 bases, 16s rRNA around 1500 bases and 23s rRNA around 2700 bases in length.

The conservation of RNA shape also makes it easier to examine the evolution of ribosomal RNA. Ribosomes exist in all living cells, across the three domains of life: bacteria, archaea and eukaryotes. The shape of an rRNA molecule is important in the correct function of a ribosome, so it is conserved well. The exact base sequence is less important and may change. Examining the mutations in base sequence and secondary structure of each functional part helps in studying the evolution of species. A mutation that appears in a group of species is a strong indicator that those species are more closely related to each other than to species lacking that mutation. Notably, the rRNA molecules of bacteria, archaea and eucaryotes each contain distinct features specific to that domain of life, which is a core observation leading to dividing all living organisms into these three domains [WKW90].

The secondary structure of RNA can also be used in distinguishing pseudogenes from actual coding genes. A pseudogene is a section of DNA that would appear to code a protein or RNA, but does not actually do so. They may be, for example, partial copies of coding genes or genes that have mutated so that they do not serve their original function any longer. While a pseudogene does not function as a gene, it may have other purposes to exist in the DNA, such as to regulate gene expression. For example, human DNA contains an estimated 1300 genes that code tRNA and several pseudogenes that have highly similar base sequences. Modeling the secondary structure helps to tell apart the genes from the pseudogenes. In order for a gene to produce functional tRNA, the RNA copied from the DNA must fold into the correct cloverleaf shape. A pseudogene, on the other hand, may produce RNA that folds into a completely different shape [LE97, LC04].

Nucleic acid design is another research topic that needs to accurately predict the RNA secondary structure. Here, artificial RNA strands with desired shape are designed and synthetized [DLWP04]. As an example, an artificial RNA strand may act as *molecular switch*, which can be used to detect the precence of tiny quantities

of other molecules [SB99].

3 Algorithmic background

In this section we will discuss different existing RNA secondary structure prediction algorithms that are interesting as background for the new safe and complete algorithm presented in this thesis. This consists of two major topics: existing safe and complete algorithms and existing RNA secondary structure prediction algorithms. We further divide the latter topic into multiple areas. First we look into different prediction methods and how multiple optimal results are handled within them. Then we discuss how the dynamic programming methods fill their dynamic programming tables in different ways, but end up with the same result. Finally, we discuss various ways to represent a single folding in both computer-readable and human-friendly formats and how to visualize multiple foldings.

3.1 Notation

There is some notation that is used often in this thesis. To avoid unnecessary repetition, the common forms of notation are introduced here.

The variable n is used to refer to the length of the RNA sequence.

References in the form of "base i" mean the base at index i within the RNA sequence.

We use (i, j) for two purposes. When denoting a base pair, it means pairing the base at index i to the base at index j. It can also denote a subsequence from base i to base j, both ends inclusive. When denoting a pair, it is assumed that $0 \leq i < j < n$. When denoting subsequences, typically $0 \leq i \leq j < n$ but in some uses $i = j + 1$ is allowed and is used to denote a zero-length subsequence. Any indices given are zero-based.

Two important dynamic programming tables appear within existing algorithms:

$V(i, j)$ contains the optimal score for the subsequence (i, j)

$V'(i, j)$ contains the optimal score for all parts of an RNA sequence outside the subsequence (i, j)

The new safe and complete algorithm uses several additional tables. Those are described in Section 4.6.

3.2 Existing safe and complete algorithms for other problems

An algorithm is *safe* if it finds parts of a solution that appear in all good solutions and *complete* if it finds all such parts.

The concept of safe and complete algorithm has been applied to assembly of DNA sequences. Here the input consists of short snippets of DNA, *reads*, as generated by DNA sequencing devices. The reads have varying amounts of overlap with each other, and by examining the overlapping parts it is possible to combine the reads into longer sequences of DNA. These generated sequences are called *contigs*. The safe and complete contig assembly algorithm finds all contigs that must have appeared in the original DNA sequence, and gives a guarantee that no more can be found from the input data [TM17].

Another safe and complete algorithm finds a sequence that could fill a gap in genome based on reads from that genome and an estimate of how long the gap might be. This algorithm finds a possible solution and indicates all parts of it that are common to all possible solutions [ST18].

3.3 Existing RNA secondary structure prediction methods

There are a large number of methods for predicting RNA secondary structure. In this review, no attempt is made to comprehensively cover all different methods. Instead, the focus is on prediction methods that are important as background to the new algorithm presented in this thesis.

The Nussinov folding algorithm [NJ80] is a dynamic programming algorithm. It computes a single folding with the maximum number of base pairs and cannot predict pseudoknots. It requires $O(n^3)$ time and $O(n^2)$ space for an RNA sequence of n bases. This algorithm is simple to implement and thus interesting as a base for the new safe and complete algorithm.

Maximizing the number of pairs is not a particularly good model of RNA folding. A better approach is a *Minimum Free Energy*, or *MFE* model, where the binding energy of the folded RNA molecule is predicted. These models consider many factors beyond the single pairings, such as how stems are more stable than the single pairs that they are made of or how loops are most stable when they are short but not too short.

Extending the Nussinov approach from simple maximization of base pairs into MFE model, there is the Zuker algorithm [Zuk89]. It uses four dynamic programming tables to compute the free energy of four different kinds of structures that may appear in the secondary structure. This algorithms makes no attempt to predict pseudoknots. For a sequence of n bases, the basic approach uses $O(n^2)$ space but exponential time. With some assumptions about the energy function, it is possible to bring the time back down to $O(n^3)$ but this further complicates the algorithm.

The Zuker algorithm is used in the popular *mfold* package [ZMT99] and the corresponding website [Zuk03]. However, while the base algorithm is essentially the same, the free energy model in this algorithm has been further refined as better knowledge of energy contribution of different RNA structures has become available [WTK$^+$94, MSZT99].

The Akutsu's algorithm [Aku00] is a dynamic programming approach that allows a limited form of pseudoknots. In its simple form this algorithm finds the maximum number of base pairs, though an extension to free energy model is also given. The simple form runs in $O(n^4)$ time for a sequence of n bases.

The Nussinov algorithm generates only a single secondary structure having the maximum number of pairs [NJ80]. The Wuchty algorithm [WFHS99] extends the Nussinov algorithm with a method that can be used to find all optimal solutions in a dynamic programming table [WB85]. The resulting algorithm generates all foldings within a defined score from the optimal solution. Wuchty et al. present two versions of this algorithm [WFHS99]. Direct extension of the Nussinov algorithm serves as an introduction to the core additions that allow generating all solutions. This is further extended to compute minimum free energy instead of maximum pairs, to turn it into a practical algorithm for predicting RNA structures. The minimum free energy version is included in the *RNAsubopt* program contained in the *ViennaRNA* package [LBH$^+$11].

None of the presented models attempt to predict preudoknots in the general case: they either forbid pseudoknots completely or limit the forms they may take. There are other algorithms that attempt to predict structures with pseudoknots, but they lose on some other desirable property. In fact, predicting pseudoknot structures in the general case is an NP-hard task [Aku00].

3.4 Existing methods for handling multiple RNA foldings

The existence of multiple good answers with the same or nearly the same score as the best answer is well-known. The existing systems tend to come with some method of computing and visualizing the different possible answers.

There are multiple methods to determine what exactly is a good result that should be found. Due to its discrete nature, maximizing the number of pairs lends itself well to simply finding all secondary structures that have exactly the best score. As minimum energy models have a more continuous scoring, this simple approach is not so useful. It is straightforward to extend finding foldings with the best score into finding all foldings that are no more than a fixed amount away from the optimal score. In free energy models, if the model represented the nature well enough, a fixed 3 kcal/mol difference from optimal score would suffice to capture all structures that occur 99.5% of the time [Zuk89].

The concept of *P-optimal* folding extends upon finding foldings a fixed amount away from the optimal score. A P-optimal folding has score at most P percent away from the optimal score [Zuk89]. For example, the 5-optimal foldings are no more than 5% worse than the best answer. A P-optimal base pair is one that appears in at least one P-optimal folding. Assuming a model where the largest score S_{max} is the best, P-optimal foldings have score S such that $S \geq (1 - P/100) \times S_{max}$.

A common approach to computing multiple answers is the Zuker method [Zuk89].

At the core of this method is an efficient way for computing the optimal folding when any two bases i and j are forced together. The Nussinov algorithm [NJ80] computes a single dynamic programming table V where $V(i, j)$ contains the maximum number of pairs for the subsequence (i, j). The Zuker algorithm adds a secondary dynamic programming table V' where $V'(i, j)$ contains the maximum score for the parts of the sequence outside the (i, j) subsequence. This new table can be computed in a similar way as the original table and with same time and space complexity. With these two tables, $V(i, j) + V'(i, j)$ is the maximum number of pairs when i and j are joined together.

A notable feature of computing a solution for each possible base pair is that it produces a single example for each. An RNA sequence can contain multiple regions that have multiple different solutions, but are independent of each other. In such situation, this approach will not attempt to evaluate all possible combinations of the partial solutions [Zuk89]. Since this method only evaluates solutions that have a single base pair choice different from default, it may not find all P-optimal solutions. At the same time, it may be easier for the user to not have to go through all combinations of the partial solutions.

The bases `NNAAANNNUUNNNNGGNNNCCCNN` can be folded in nine different ways, each of which contains four base pairs. These foldings are shown in Figure 6. This example uses a non-pairing base `N` to make the example straightforward to understand. They could be replaced with standard bases without changing the results. Since this example has two regions with two pairing choices each, the Zuker method finds only four foldings: when a pair in one region is forced, the other region receives the default folding. It should be noted that this default folding is arbitrary and depends on implementation details of the dynamic programming backtrack algorithm.

The Zuker method can generate at most one folding per a base pair, giving at most $(n \times (n - 1))/2$ different foldings for an RNA sequence of n bases. The number of optimal solutions, however, can be exponential in the number of bases as shown in Section 4.5. At fairly low sequence lengths the number of possible foldings goes over the number of pairs. Thus, by pigeonhole principle, there are inputs where the Zuker method cannot find all optimal foldings.

While the method of forcing all base pairs and computing the score of each does not give all possible foldings, a study made with tRNA^{phe} from yeast (EMBL database nr. RF6280 [SSHS96]) found that the method does find a fairly representative sample of all solutions [WFHS99].

Instead of sampling the optimal solutions by forcing each base pair, we can employ the Wuchty algorithm to generate all solutions within a specified score interval from the optimal solution [WFHS99]. Finding all solutions guarantees that the sampling does not lose any interesting solutions, though it is unclear if this is actually a concern.

Finding all near-optimal solutions with Wuchty algorithm may yield a large number of foldings, so a method for effectively handling and visualizing them is even more important than with the Zuker algorithm.

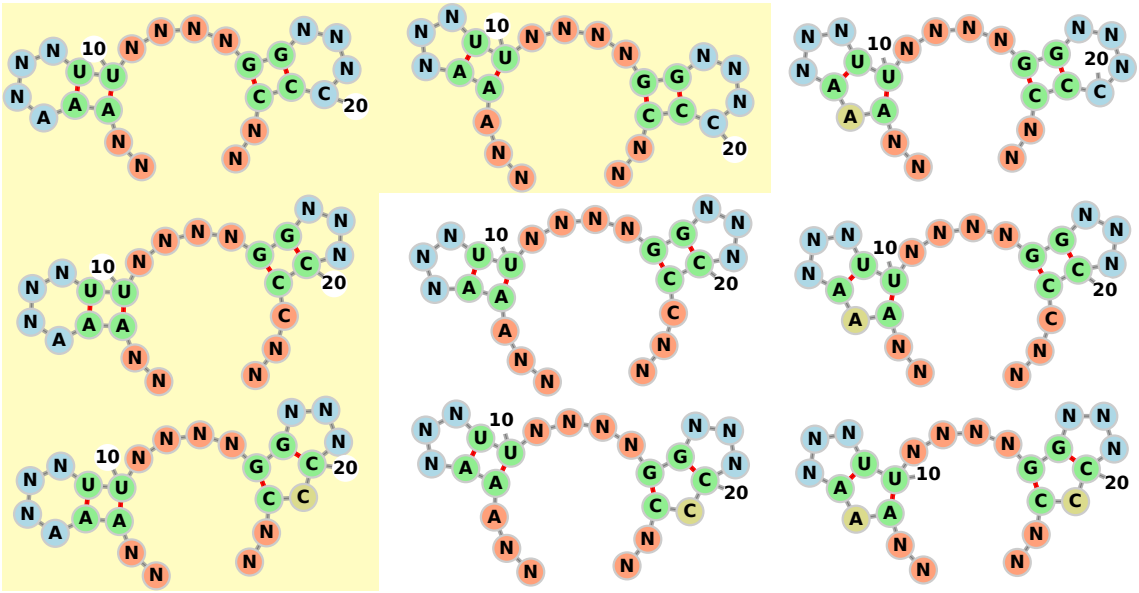


Figure 6: All possible foldings for a simple example. Foldings that are found with the Zuker method are highlighted.

3.5 Different formulations for filling the dynamic programming table

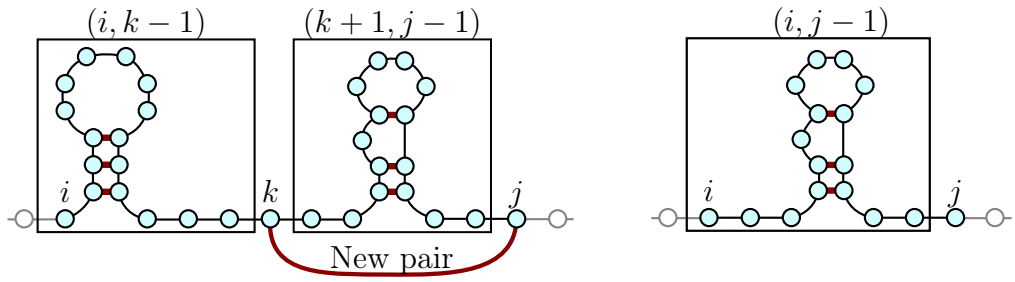
There exist multiple possible formulations for the recurrence relation that is used to fill the dynamic programming table. All of these, however, share common base case. All cells $V(i, i)$ of the dynamic programming table represent one-length subsequences. A single base can not form pairs, so $V(i, i) = 0$ for all $0 \leq i < n$. Further left in the table are cells $V(i-1, i)$ which represent zero-length sequences and are also defined with $V(i-1, i) = 0$ for all $0 < i < n$. Cells $V(i, j)$ where $i - 1 > j$ are not used.

The original Nussinov algorithm uses two cases [NJ80]:

$$V(i,j) = \max \begin{cases} \max \big\{ V(i, k-1) + V(k+1, j-1) + 1 \;\big|\; \\ \qquad i \leq k < j \text{ and bases } k \text{ and } j \text{ can be paired} \big\}, \\ V(i, j-1) \end{cases}$$

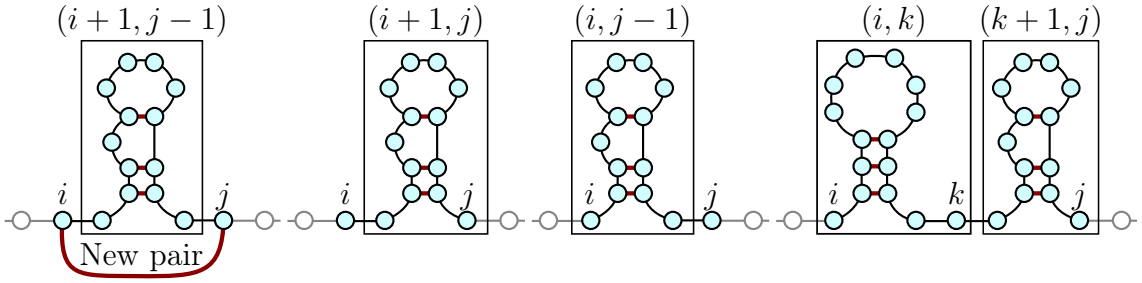
These cases are also shown in Figure 7. In the first case the Nussinov algorithm attempts to pair any base j in range from i to $k-1$ with the base k. Whenever the pairing is possible, it uses the previously computed optimal foldings for the shorter subsequences $(i, k-1)$ and $(k+1, j-1)$ to compute the new optimal folding. The second case allows for non-paired bases. In it, the algorithm takes a previously computed subsequence $(i, j-1)$ and appends the base j to it as a non-paired base.

The original two cases may not be the clearest to understand. The same functionality could be described as four possible cases [Edd04]. The subsequence (i, j) could be



(a) Pair k and j (b) Append j as unpaired

Figure 7: Cases in the Nussinov algorithm



1. Pair i and j 2. Prefix i 3. Suffix j 4. Join (i, k) and $(k+1, j)$

Figure 8: Four cases for filling the dynamic programming table [Edd04]

formed by:

1. If bases i and j can form a pair, taking the sequence for $(i+1, j-1)$ and adding the base pair.
2. Taking $(i+1, j)$ and prefixing i as an unpaired base.
3. Taking $(i, j-1)$ and suffixing j as an unpaired base.
4. Joining two subsequences (i, k) and $(k+1, j)$ where $i \leq k < j$.

These four cases are also shown in Figure 8.

Only two of these cases are actually needed. The case 2, computing (i, j) by prefixing unpaired base i to $(i+1, j)$, is same as joining (i, i) to $(i+1, j)$, which happens in case 4 with the choice $k = i$. Similarly the case 3, computing (i, j) by suffixing unpaired base j to $(i, j-1)$, is same as joining $(i, j-1)$ to (j, j) which occurs in case 4 with choice $k = j-1$.

This observation leads to another published version of the dynamic programming recurrence [Sun10, chapter 11.5]. Eliminating the two redundant cases leaves us with a version with only two possible cases:

$$V(i,j) = \max \begin{cases} V(i+1, j-1) + \delta(i,j) & \text{pair bases } i \text{ and } j \\ \max_{i \leq k < j}(V(i,k) + V(k+1,j)) & \text{combine two subsolutions} \end{cases}$$

```
>RS1661 GCU E.COLI
GGUGAGGNGGCCGAGAGGCNGAAGGCGCUCCCNUGCUNAGGGAGUAUGCGGUCAAAAGCUGCAUCCGGGGUUCGAAUCCCCGCCUCACCGCCA
(((((((..(((..........)))((((((.......))))))..................(((((.......)))))))))))....
```

Figure 9: FASTA file combined with a folding in dot-bracket format

Here $\delta(i, j)$ is the score for joining bases i and j. In the simplest formulation, it is 1 for Watson-Crick base pairs and wobble pairs and 0 otherwise. This formulation does imply that pairing bases i and j that cannot be paired is equivalent to taking subsequence $(i + 1, j - 1)$ and adding the bases to it as unpaired. While this is true when counting number of base pairs in optimal solution, many future uses need to distinguish between the two. In them, creating a pair should only be done when an actual pair can be formed.

3.6 Representations of an RNA folding

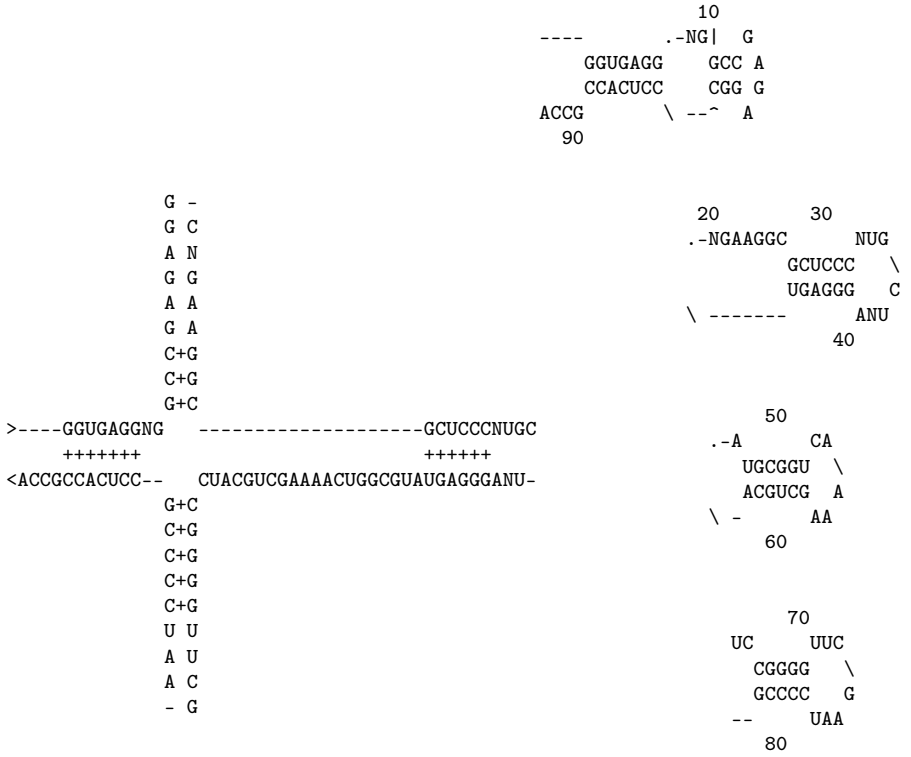
In order to predict a folding, a sequence is first needed, and that sequence needs to be represented somehow. A common sequence format is a *FASTA* file. Such file essentially consists of a comment line describing the sequence and of a line containing the bases of the sequence as single-character identifiers. This format is further discussed in Section 5.1.

A common representation of a folding is the dot-bracket notation. A dot represents a free base and opening and closing brackets are used to represent the $5'$ end and the $3'$ end of a base pair. This notation is often combined with FASTA sequence format, giving a file that contains both the base sequence and how it is folded. Figure 9 shows an example of a sequence in this format.

When there are no pseudoknots, the dot-bracket notation is unambigious. In order to support pseudoknots, this format must be extended. The *RNA STRAND* database [ABHC08] uses different types of brackets `()`, `[]`, `{}` and `<>` and upper- and lowercase letters to represent the beginning and end of pairs.

While the dot-bracket format is a simple and concise format, more feature-rich formats are sometimes called for. One of them is the `.ct` format produced by the mfold package [Zuk03]. This is a simple text-based format, where each line describes the details of a single base. A line contains six items: 1-based base index, base type as single letter, index of the previous base, index ofthe next base, index of the base to which this base is paired, or zero if none, and natural numbering for the base. With this structure, representing pseudoknots is straightforward and unambigious. Parsing this structure to determine where each base is paired to is simpler that with dot-pair notation, which requires a stack-based parser that keeps track where each not-yet-closed bracket was. There also exists a more concise version of this format, called a `region` file, where each line gives details of a range of pairs or a range of unpaired bases.

There are attempts to create a uniform RNA information interchange format. For



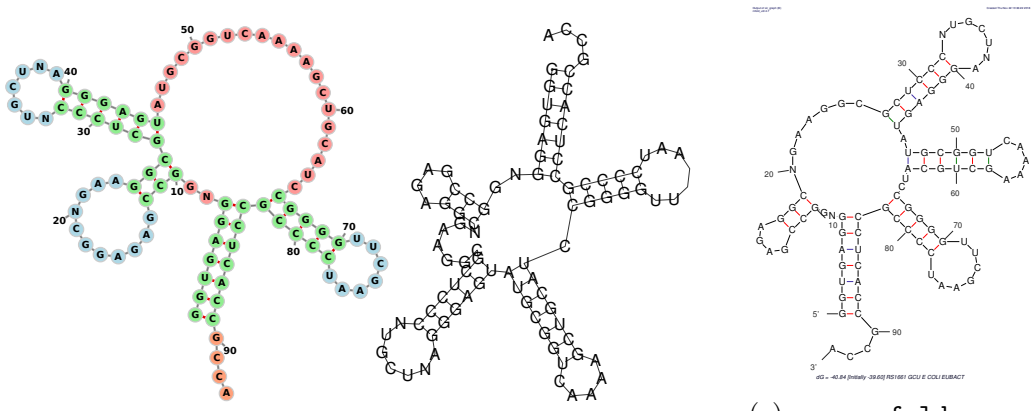
(a) From implementation made for this paper

(b) From mfold web server [Zuk03]

Figure 10: ASCII art visualizations of RNA folding (RS1661 from Sprizl database [SSHS96])

example, an XML-based format called *RNAML* [WGA+02] is supported by several packages such as RNA STRAND database [ABHC08] and mfold web server [Zuk03]. This format not only represents a sequence and its folding, but can represent foldings determined through multiple methods, tertiary structures, interactions between multiple RNA strands and various other situations.

In addition to the computer-readable formats, many tools provide human-friendly visualizations of the folding structure. Both the mfold package [Zuk03] and the `keltainen.duckdns.org/rnafolding/format` package developed for this thesis implement an ASCII art visualization. Such a visualization is good for text terminals, so a user working with the terminal can see the folding structure without opening an additional display program. Examples of such visualization are shown in Figure 11. Vector graphics formats are also popular for visualizations, as they can yield clear and crisp images that can be used both on screen and in print. For example, Figure 11a shows *SVG* format (Scalable Vector Graphics) output from *forna*, which is a web application that can display a folding completely in a web browser [KHH15]. Figure 11b shows PostScript export from the ViennaRNA package [LBH+11], which contains such output in addition to multiple methods for secondary structure prediction. Figure 11c shows a similar output from the mfold package [Zuk03].



(a) forna [KHH15] (b) ViennaRNA [LBH+11] (c) `mfold` web server [Zuk03]

Figure 11: Examples of same tRNA folding displayed with different tools

3.7 Visualization of multiple RNA foldings

The Zuker method gives all optimal or P-optimal foldings that can be produced by joining a single pair. A dot plot is a visualization of the foldings produced in this manner. In a dot plot, a dot is drawn at coordinates (i, j) whenever the pair (i, j) appears in any of the produced P-optimal foldings [Zuk89].

An example of a dot plot appears in Figure 12. This figure represents all Zuker foldings within 5 kcal/mol of the optimal folding that has energy -22.10 kcal/mol, as generated with the RNAsubopt program from the ViennaRNA package [LBH+11]. The large diagonal line from top left to bottom right represents free bases that appear in any of the foldings. In its original form, a dot plot does not contain such diagonal, but only shows the possible base pairs.

The dot plot in Figure 12 shows some bottom left to top right diagonal lines. When alone, such line represents a stem structure that appears in all predicted structures. The top right corner of this example shows multiple diagonal lines grouped closely together. This represents a case where approximately the same bases pair to form a stem, but the exact configuration varies between different predictions.

Originally the dot plot was not only used for visualization of different foldings, but as part of an user interface. The user could choose any pair shown in the plot to receive an example P-optimal folding that includes the selected pair [Zuk89].

With an algorithm that can compute binding probabilities of each pair, a superficially similar *box matrix* plot can be created [McC90]. In such plot, a point is plotted at (i, j) with point size or colour depending on the probability of bases (i, j) forming a pair. This can give a more fine-grained representation of possible structures than a dot plot. The MFE algorithms may give very few near-optimal structures overall and where variation exists, a dot plot treats all variants as equal. The box matrix plots introduced by McCaskill [McC90] use base-10 logarithmic scale for the probability so that the plots can show a huge scale of possible probabilities.

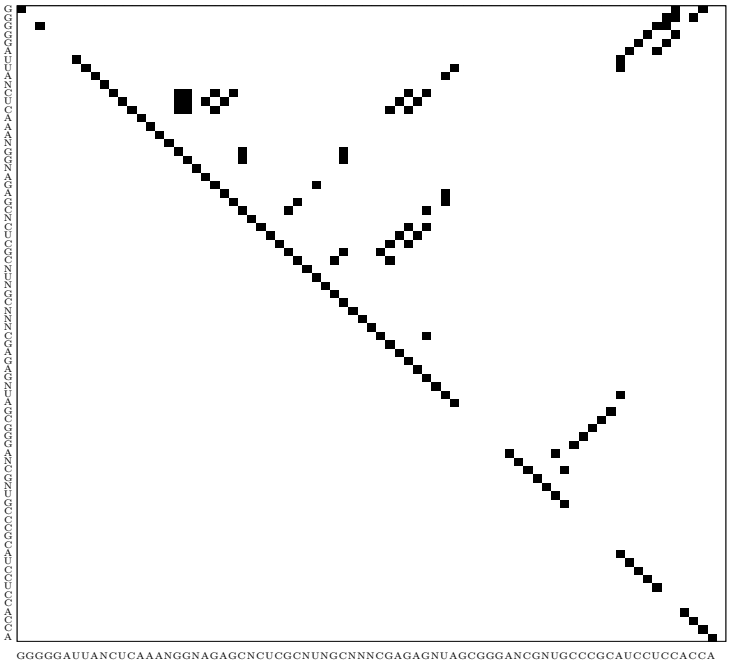


Figure 12: Dot plot for MFE Zuker pairings within 5 kcal/mol of optimal for human tRNA (RA9990 in Sprinzl database [SSHS96])

4 Safe and complete algorithm for RNA secondary structure prediction

Both RNA secondary structure prediction algorithms and safe and complete algorithms have been presented in earlier work. Combining the two approaches, however, is non-trivial. A novel dynamic programming algorithm that combines RNA secondary structure prediction with the safe and complete approach is presented in this section.

We begin by examining the background upon which the algorithms is built. In Section 4.5 we show a simple but slow safe and complete algorithm. The Sections 4.6 through 4.9 describe a more efficient dynamic programming algorithm. We also show a solution tree data structure, in Section 4.10, which can be used to generate the full set of optimal solutions when needed. Finally, the Section 4.11 discusses cases where they may be multiple distinct ways to produce the exact same folding and how to deal with those.

4.1 Base algorithm

There are multiple possible dynamic programming algorithms that could be used as base for the new algorithm.

The simplest starting point would be the Nussinov algorithm [NJ80] for computing

maximal number of pairs.

Zuker algorithm [Zuk89] contains all the details needed for computing good estimates of RNA molecule free energy. Owing to the complexity of using four different tables and having a complex method for filling them, extending this algorithm would be more complex than extending the Nussinov algorithm. However, the current version of the Zuker algorithm is state-of-the-art in terms of how well the predictions match the biological ground truth. Extending the Zuker algorithm would permit direct comparisons of the biological correctness between different modern algorithms and the new algorithm.

Akutsu's algorithm [Aku00] is also a potential starting point for the extended algorithm and would allow predicting pseudoknot structures. It is more complex than the Nussinov algorithm, but not massively so. The higher time complexity of $O(n^4)$ might prove troublesome when predicting the secondary structure of large RNA sequences.

The Wuchty algorithm [WFHS99] predicts all solutions within a specified score interval from optimal solution. There exists both a version that predicts maximum number of base pairs and a version that predicts minimal free energy. As this algorithm gives all solutions, it is rather close to the new algorithm presented here. The backtrack phase, however, uses a significantly different approach. By not choosing this algorithm as base, we have managed to develop this new approach with different performance characteristics.

The Nussinov algorithm is chosen as the base for the new algorithm due to its simplicity. The Wuchty algorithm serves as an excellent comparison point during the implementation since it generates all solutions, like the new algorithm, but through a different method. This allows cross-verification of the two algorithms, so it is possible to be reasonably sure that the implementation is correct.

The algorithm description presented here is based on the Nussinov algorithm [NJ80]. It should be noted that the algorithm implementation uses a recurrence relation from [Sun10] combined with the degenerate case elimination method described in Section 4.11. Crucially, both of these approaches yield the same results but the recurrence relations of Nussinov algorithm make it significantly simpler to describe the algorithm. Both of these recurrence relations were described in more detail in Section 3.5.

When implementing a Minimum Free Energy algorithm, it is useful to implement the lookup of P-optimal foldings which are almost as good as the strictly optimal foldings. However, the algorithm implementation in this thesis finds only the strictly optimal ones.

4.2 Structure of a solution within the dynamic programming table

Earlier safe and complete solutions [TM17, ST18] have been built for cases where a solution is represented by a path in the dynamic programming table. However, this is not the case for RNA secondary structure prediction. Instead, a single solution becomes a tree in the dynamic programming table. Extending the safe and complete approach into tree-like solutions presents a novel challenge. Here we will show that the structure of a solution is indeed a tree.

Theorem 1. *A single folding is represented by a tree in the dynamic programming table.*

Proof. The proof is in two parts. First we will show that a tree is a necessary structure and a path would be insufficient. Then we will show that a tree is a sufficient structure, and a graph is not necessary.

There are two cases in the Nussinov recurrence relations. The second case, suffixing a free base into a subsolution, can only produce a diagonal path in the dynamic programming table. The first case, however, produces a binary tree structure as it joins two subsolutions into one and both of these subsolutions may also have been produced by joining two subsolutions. This shows that a tree is a necessary structure.

To show that a tree is sufficient structure and a graph is not necessary, we will examine the areas of the dynamic programming table that can be used to construct each subsolution. A graph might be needed if there can be some cell (i, j) of the dynamic programming table that is produced by joining two subsolutions and another cell (a, b) that is used by both subsolutions.

An optimal solution for any subsequence (i, j) is built on solutions with at least as large i component and at most as large j. That is, the solution for (k, l) can be a part of a solution for (i, j) only when $k \geq i$ and $l \leq j$.

A solution for (i, j) may be produced by joining subsolutions $(i, k-1)$ and $(k+1, j-1)$, $i \leq k < j$. If these parts contained a common part (a, b), it would hence hold that $a \geq k+1$ and $b \leq k-1$. In order for (a, b) to be within the dynamic programming table, $a - 1 \leq b$. From these it can be derived $a - 1 \leq b \leq k - 1$ and $a - 1 \geq k$. These two can never hold simultaneously, so this common part (a, b) cannot exist within the dynamic programming table. □

As was shown above, the two parts of a join, $(i, k-1)$ and $(k+1, j-1)$, $i \leq k < j$, are always built from disjoint parts of the dynamic programming table. These two parts can be seen in Figure 13. The top right half is the part of dynamic programming table that represents non-empty subsequences and is actually used in the algorithm. The shaded diagonal, where $i - 1 = j$ may be read but these cells represent empty subsequences and always contains value zero. The gray area in bottom left is never used. The value in any cell of the dynamic programming table is constructed using

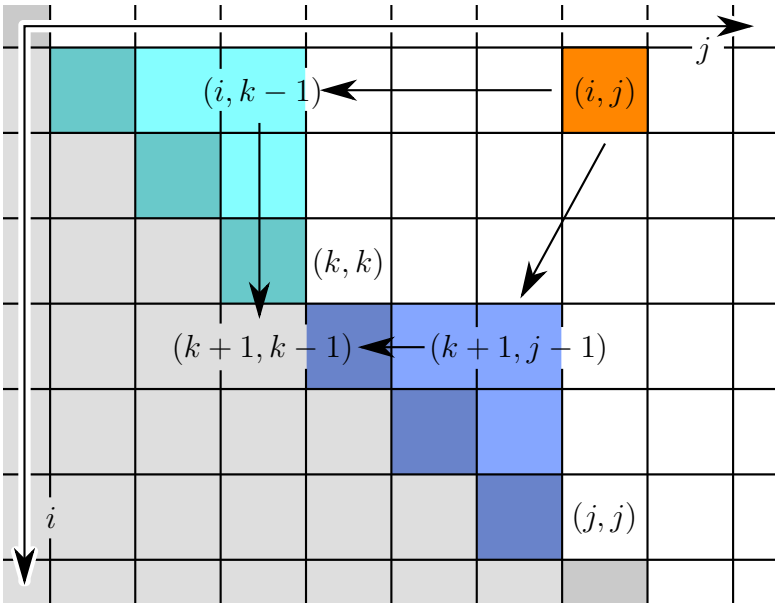


Figure 13: When combining two subsolutions, the subsolutions are disjoint

cells left and down from it. The areas that may be used to construct $(i, k-1)$ and $(k+1, j-1)$ only cross in the gray area that is never used within the algorithm.

4.3 Backtracking

The backtracking phase of a simple dynamic programming algorithm uses the dynamic programming table generated using rules shown in Section 3.5 and traces a single tree from the answer to the base cases at table diagonal. To extend this into a safe and complete algorithm, we would trace back not just a single answer, but all answers that would produce an equally good result.

Looking at a single cell, the default backtrack algorithm would find one way how the score in that cell could have been generated, whereas the new algorithm finds all such ways and traces back each of them. Conceptually this is highly similar to the Wuchty all-solutions algorithm [WFHS99], but the implementation is significantly different.

The Wuchty algorithm uses a stack $\mathscr{S}$ of algorithm states. Each algorithm state consists of a stack σ of subsequences that have not yet been explored and a list $\mathscr{P}$ of pairs that belong to the folding. On each round of the algorithm, a subsequence (i, j) is popped from stack σ. The shorter subsequences that it is made of are pushed back to σ and, if the construction pairs some bases, the pair is pushed to $\mathscr{P}$. When there are k ways to produce (i, j), this representation of algorithm state allows making k copies of the current state. The σ and $\mathscr{P}$ of each state copy are updated with the details of the corresponding way of producing (i, j). Each of these new states is then pushed onto the state stack $\mathscr{S}$. When the stack σ of one state is empty, the list $\mathscr{P}$ contains all pairs of one folding. Then, the algorithm can move onward to process

the next item in state stack $\mathscr{S}$. Once the state stack is empty, all possible foldings have been generated.

Unlike the Wuchty all-solutions algorithm, the backtracking algorithm presented here does not produce an actual list of all possible solutions. Rather, it can produce different counts for foldings, as described in Section 4.6, and a tree structure representing how all possible solutions are constructed, which is shown in Section 4.10. Either of these can be produced independently of the other.

The backtrack algorithm uses two cases for backtracking from cell (i, j):

Pairing Backtrack to all divisions to $(i, k-1)$ and $(k+1, j-1)$, $i \leq k < j$ where (k, j) can form a pair and $V(i, k-1) + V(k+1, j-1) + 1 = V(i, j)$.

Free base Backtrack to $(i, j-1)$ if $V(i, j-1) = V(i, j)$.

4.4 Number of possible foldings

The time and space complexity of algorithms presented here depends on the number of foldings. In order to properly analyze these algorithms, we need to know how many foldings a folding predictor algorithm might produce. It is known that the number of optimal or near-optimal solutions is roughly exponential in n, the number of bases in the sequence. The experimental analysis with natural RNA sequences in Section 6.1 also shows this exponential behaviour. Also, when near-optimal solutions are computed, increasing the range of admitted solutions increases the number of foldings roughly exponentially [WFHS99].

As a rough explanation why the number of optimal foldings is necessarily exponential, let us consider a short RNA sequence that has two possible foldings with equal number of pairs. Concatenating k such sequences into a larger one lets us choose one of the two alternatives for each of the k short sequences, giving a total of 2^k possible solutions.

We demonstrate that there may be an exponential number of foldings. This is done through two theorems. The first theorem shows that there are sequences that have an exponential number of optimal foldings. The second one shows that there cannot be more than 3^n foldings when pseudoknots are not allowed.

Theorem 2. *There are RNA sequences of n bases that have $C_{n/2}$ distinct optimal foldings, where C_i denotes the Catalan number. This number is exponential in n.*

Proof. We will examine an RNA sequence consisting of repeating bases `AU`. We will only consider sequences whose length n is even. For the purposes of this proof, we will attempt to predict a maximum pairing with no further limitations. Notably, the minimum hairpin size limit described in Section 5.3 is not used here.

In this sequence, base i is `A` if i is even and `U` if i is odd. Folding the sequence in half, the base i pairs with base $n - i - 1$. Since n is even, it is easy to see that whenever

i is even, $n-i-1$ is odd and vise versa. It follows that folding the sequence in half creates a folding where each of the bases is paired. Thus all optimal foldings of this sequence pair all bases.

Let us present an optimal folding of this sequence in dot-bracket notation. As all bases are paired, a folding without pseudoknots consists only of brackets and those brackets are balanced. Thus the folding, presented in dot-bracket format, is a *Dyck word*.

All Dyck words are also optimal foldings of the sequence. In order for an opening bracket and a closing bracket to match, there must be an equal number of opening brackets and closing brackets between them. In other words, between matching brackets there is always an even number of brackets. This means that one of the matching brackets is at even index, so it corresponds to base `A`, and the other is at odd index, corresponding to base `U`.

The number of Dyck words of length n is equal to the *Catalan number* $C_{n/2}$ [Sta99]. Thus the number of possible foldings for our sequence is also $C_{n/2}$.

In order to further analyze the number of sequences, let us use an asymptotic approximation:

$$\begin{aligned} C_{n/2} &\sim \frac{4^{n/2}}{(n/2)^{3/2}\sqrt{\pi}} = \frac{\sqrt{4}^{\,n}}{\sqrt{n^3/2^3 \times \pi}} = \frac{2^n}{\sqrt{n^3}\sqrt{8 \times \pi}} \\ &= \Theta\left(\frac{2^n}{n^{3/2}}\right) \end{aligned}$$

While the asymptotic approximation is not purely exponential, it is clear to see that the exponential numerator grows much faster than the polynomial denumerator. Therefore, this derived number can be considered exponential in n. □

Theorem 3. *RNA sequence of n bases can have no more than 3^n distinct pseudoknot-free foldings.*

Proof. A folding without pseudoknots can be written in dot-bracket notation using only three characters, $\cdot$, `(` and `)`. Dot-bracket format is discussed in further detail in Section 3.6. Such representation of a valid folding is unique and any sequence of these characters either represents an unique folding or does not represent any folding.

When there is a sequence of n bases, any of its foldings can be uniquely represented with n-length sequence of these three characters. As there are 3^n such sequences, there cannot exist more than 3^n possible foldings. This upper limit is not strict, since many such sequences — such as `(((((` or $\cdot\cdot)\cdot\cdot$ — do not represent a folding. □

The algorithm presented in this thesis is a maximum pairs algorithm that finds all possible solutions, so it must produce exactly the same solutions as Wuchty algorithm would. It also produces only solutions without pseudoknots. With these

we have shown that this new algorithm may produce an exponential number, but no more than 3^n, distinct foldings. Importantly for some of the analysis in following sections, this means that the number of foldings can be represented with a number containing $O(n)$ bits.

4.5 Trivial algorithm for computing safety

In cases where the complete set of allowed solutions is generated, computing the safety of each base is straightforward. The solutions are checked one base at a time. If a given base is unpaired in each solution or is paired to the same base in each solution, that base is safe.

In a similar way, it is possible to create a free base count array and a pairing table. Cell $G(i)$ of the base count array is set to the number of solutions, in which base i is free. Cell $R(i, j)$, $i < j$ in pairing table is set to the number of solutions, where base i is paired to base j.

This trivial algorithm takes $O(sn + n^2)$ time. The sn term appears since this algorithm goes over s solutions and checks the n bases for each. The n^2 term appears since the pair count table R contains $O(n^2)$ elements and each must be initialized. It should be noted that while this running time is polynomial in the number of results, it can be exponential in the number of bases. This happens because s may be exponential in n, as was discussed in Section 4.4.

4.6 Dynamic programming algorithm for safety

The trivial algorithm for determining safety has exponential time complexity in the number of input bases, so it becomes prohibitively expensive to run with larger RNA samples. A completely different approach is needed to efficiently cope with the long sequences. Here we present a novel algorithm for this task.

The new algorithm applies dynamic programming method to derive the number of solutions that contain any given pair or any given unpaired base. This algorithm is composed of three distinct steps:

1. Compute the number of optimal foldings for any subsequence, creating table S (Section 4.7)
2. Compute the number of optimal foldings for the full sequence that use any given subsequence, creating table T (Section 4.8)
3. Compute the number of optimal foldings that use any base pair or any base unpaired, giving array G for the unpaired bases and table R for the pairs (Section 4.9)

As a prerequisite for this algorithm, the filled dynamic programming table V for optimal folding score must be available to use.

It is important to distinguish two similar concepts in the following description. For a given subsequence (i, j) we have the number of possible ways to generate the optimal solution for it, $S(i, j)$. We also have the number of optimal solutions for the full sequence that use those optimal solutions for the subsequence, $T(i, j)$. As an example, let us consider a single base i in the sequence. There is exactly one way to create an optimal folding for that single base, shown in solution count table as $S(i, i) = 1$. However, any number of optimal solutions may use that single base as unpaired base in them, hence $0 \leq T(i, i) \leq S(0, n - 1)$.

When analyzing the time complexity of this algorithm, we have to take into account that the number of foldings may be exponential in input size. As this algorithm needs to store and perform arithmetic on the number of foldings, we cannot assume constant-time integer arithmetic. As discussed in Section 4.4, a sequence of n bases may have at most 3^n foldings. Representing such number takes $\log_2(3^n) = O(n)$ bits. As the algorithms need to perform multiplications and divisions and there exists a wide variety of multiplication algorithms, let us denote time of multiplying or dividing two n-bit values with $M(n)$. For a straightforward algorithm $M(n) = O(n^2)$, but other more efficient algorithms exist, too.

This algorithm takes $O(n^3 M(n))$ time and $O(n^3)$ space. This time efficiency is polynomial, which is already a great improvement over the trivial algorithm. The details of the time complexity will be discussed in detail for each part of the algorithm in the corresponding subsection.

4.7 Counting solutions

The number of optimal solutions can be computed using the dynamic programming method. This method is highly similar to how the optimal number of base pairs is computed. It uses the dynamic programming table V that contains the optimal number of pairs for any subsequence (i, j).

Let us name the subsolution count table S. The aim is to fill this table such that $S(i, j)$ contains the total number of distinct optimal foldings for subsequence (i, j). In other words, $S(i, j)$ is the number of distinct foldings for (i, j) that contain $V(i, j)$ pairs.

To create a base case, any sequence containing at most one base clearly has exactly one optimal folding. In other words, $S(i, j) = 1$ when $i = j$ or $i - 1 = j$.

The solution counting uses the same two cases as the backtrack algorithm described in Section 4.3.

In the pairing case, joining two subsequences $(i, k - 1)$ and $(k + 1, j - 1)$ produces an optimal solution for sequence (i, j). Every combination of optimal solutions to the first and the second part will produce a unique optimal solution to the larger subsequence. Such combinations together contribute $S(i, k - 1) \times S(k + 1, j - 1)$ possible optimal solutions to the new subsequence.

In the free base case the subsequence (i, j) is produced by appending a single un-

paired base to $(i, j-1)$. In this case, the longer subsequence can be produced in $S(i, j-1)$ ways.

There often exists multiple different ways to create an optimal solution for given (i, j) subsequence. When there is, the number of ways in which each solution can be produced is summed up to produce the total number.

```
for each subsequence (i, j) in increasing order of length do
    s ← 0
    for each optimal way of constructing (i, j) from shorter subsequences do
        if construction joins subsequences (i, k − 1) and (k + 1, j − 1) then
            s ← s + S(i, k − 1) × S(k + 1, j − 1)
        else                          ▷ construction uses single subsequence (i, j − 1)
            s ← s + S(i, j − 1)
        end if
    end for
    S(i, j) ← n
end for
```

The time complexity of this algorithm is $O(n^3 M(n))$. It goes over $O(n^2)$ subsequences, tries $O(n)$ possible ways to construct each and performs a multiplication requiring $O(M(n))$ time for each.

4.8 Counting optimal foldings containing a specific subsequence

We will compute a new table T where $T(i, j)$ contains the number of optimal foldings for the full sequence that use an optimal folding for subsequence (i, j). This new table can be computed from the data in table S.

The table T is constructed starting from full sequence $T(0, n-1)$ and moving towards single base subsequences. It can be seen that the number of optimal solutions for the full sequence is the same as number of optimal solutions that use the full sequence. Thus, the base case for this algorithm is $T(0, n-1) = S(0, n-1)$.

Previous dynamic programming algorithms presented here have been of collecting type: the required values for computing each dynamic programming cell have been computed earlier in the algorithm. The value for current cell is computed from earlier results using a recurrence function. This algorithm will be a distributing one instead. When this algorithm enters a cell, the value of that cell has already been computed. This value is then used to refine values in cells that will be entered later.

When entering a cell (i, j), the algorithm determines all distinct ways how the subsequence (i, j) can be generated from shorter subsequences. This is done as in the backtracking algorithm described in Section 4.3.

The goal is to determine how many foldings out of $T(i, j)$ use each of the distinct ways to produce (i, j) from shorter subsequences, and to add that number to the cell of that subsequence in T. We will process the subsequences in length order

from longer to shorter. As any subsequence can only be a part of strictly longer subsequences, this ordering guarantees us that final value of T for any subsequence is available when that subsequence is processed.

The value $T(i,j)/S(i,j)$ represents the number of times the full set of optimal foldings for subsequence (i,j) is used in constructing all optimal foldings of the full sequence. It can be seen that $T(i,j)$ is always divisible by $S(i,j)$. In this algorithm, this fraction is called c or the *scaling factor*.

The value of $S(i,j)$ was constructed by summing the number of solutions of each possible way of constructing (i,j). Now, this algorithm will distribute the value $T(i,j)$ out to those subsolutions in the exact same way.

for each subsequence (i,j) in decreasing order of length **do**
  $c \leftarrow T(i,j)/S(i,j)$
  for each optimal way of constructing (i,j) from shorter subsequences **do**
    if this construction pairs k with j **then**
      $a \leftarrow c \times S(i,k-1) \times S(k+1,j-1)$
      $T(i,k-1) \leftarrow T(i,k-1) + a$
      $T(k+1,j-1) \leftarrow T(k+1,j-1) + a$
    else ▷ this construction appends j as unpaired to $(i,j-1)$
      $a \leftarrow c \times S(i,j-1)$
      $T(i,j-1) \leftarrow T(i,j-1) + a$
      $T(j,j) \leftarrow T(j,j) + a$
    end if
  end for
end for

The time complexity of this algorithm is $O(n^3 M(n))$. It goes over $O(n^2)$ subsequences. Every subsequence has $O(n)$ possible constructions. For each construction, the algorithm performs a division and a few multiplications, requiring $O(M(n))$ time.

4.9 Counting optimal foldings containing a pair or an unpaired base

An interesting property of the table T is that its diagonal values $T(i,i)$ represent how many of all optimal foldings contain that single-base sequence. In other words, in how many foldings that base appears unpaired. This is directly interesting in computing safety, so let us create a new array G out of the single-base values on the diagonal, having values $G(i) = T(i,i)$.

Deriving the pair table R is somewhat more complex. The aim is that $R(i,j)$ would contain the number of optimal foldings that use the pair (i,j). This is not directly available from T as any subsequence containing i and ending in j might add this pair, or not. However, this can be computed by going over all subsequences and determining if the subsequence directly uses that pair. Summing the number of foldings that use each of these subsequences gives us the value of R.

```
for each subsequence (i, j) do
    c ← T(i, j)/S(i, j)
    for each optimal way of constructing (i, j) from shorter subsequences do
        if this construction pairs base k with j then
            R(k, j) ← R(k, j) + c × S(i, k − 1) × S(k + 1, j − 1)
        end if
    end for
end for
```

The time complexity of this algorithm is $O(n^3 M(n))$. It goes over $O(n^2)$ subsequences and tries $O(n)$ possible ways to construct each. For each such try, the algorithm performs one or two multiplications requiring $O(M(n))$ time each.

It can be noted that this algorithm has no requirements on the order of iteration over the subsequences, and it only uses the values $T(i, j)$ and $S(i, j)$ for any given subsequence (i, j). Due to these properties, it is possible to join this and the algorithm for computing table T, so that both tables are computed in one pass.

With the tables G and R in place, it is straightforward to determine the safety of pairs and free bases in any given folding. When the folding has a pair (i, j), that pair and its bases are safe if and only if $R(i, j)$ is equal to the total number of optimal foldings $S(0, n-1)$. Similarly, a free base i is safe if and only if $G(i) = S(0, n-1)$.

4.10 Solution tree data structure

In addition to the counting algorithm presented above, we have implemented a backtracking algorithm. This algorithm does not directly produce the full list of optimal solutions, but it produces a *solution tree* data structure instead. This data structure stores all solutions without the need to evaluate each solution individually, and it is straightforward to get the individual solutions when needed.

Both the Wuchty algorithm and the solution tree can be used to generate the full set of optimal solutions, but they take significantly different approaches to do so. Both algorithms always produce the same set of foldings, which property is used in the implementation to verify correctness.

Each node of the solution tree contains a list of free bases and a list of base pairs. Additionally, they may contain either a join of two subsolutions (*join node*) or a list of possible alternative solutions (*alternatives node*). The real benefit of this structure comes when there exists a join node and both of its subtrees contain alternatives nodes. If the prefix of join has p optimal foldings and the suffix s optimal foldings, they result in $p \times s$ foldings when joined. This makes it possible to store a massive number of foldings in a relatively small space.

A simple example of a solution tree is shown in Figure 14. In this figure, there is one join node (node 1), two alternatives nodes (nodes 2 and 3) and four leaf nodes (4-7). This tree represents four foldings, each of which combines one of the two alternatives under node 2 with one of the two alternatives under node 3.

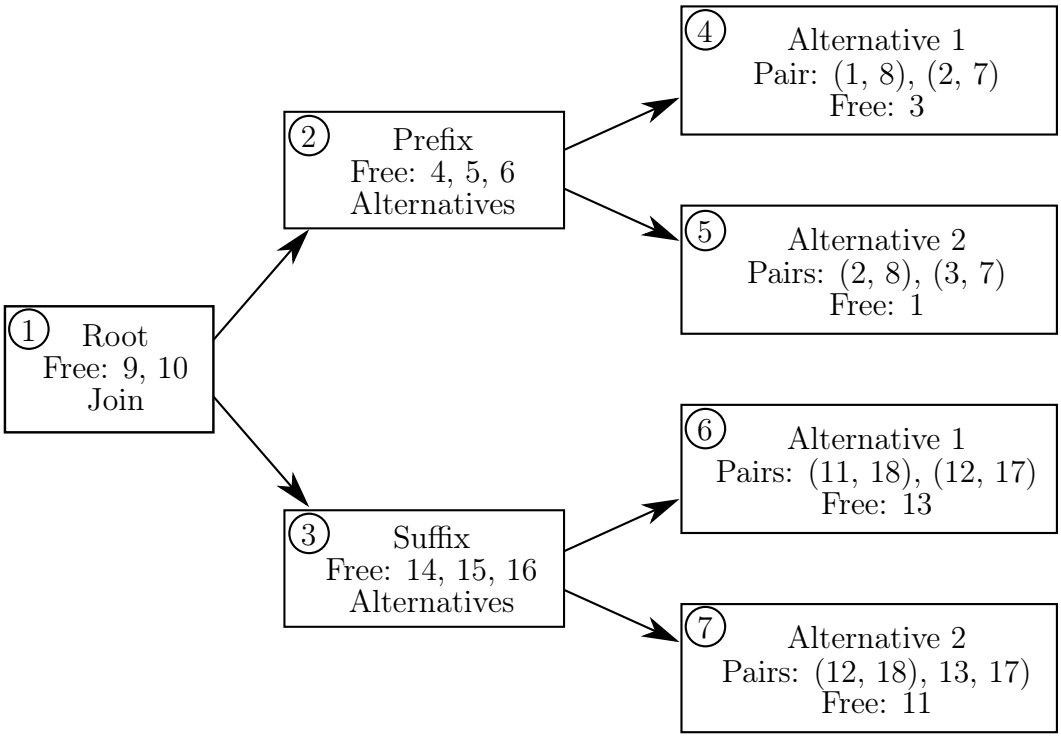
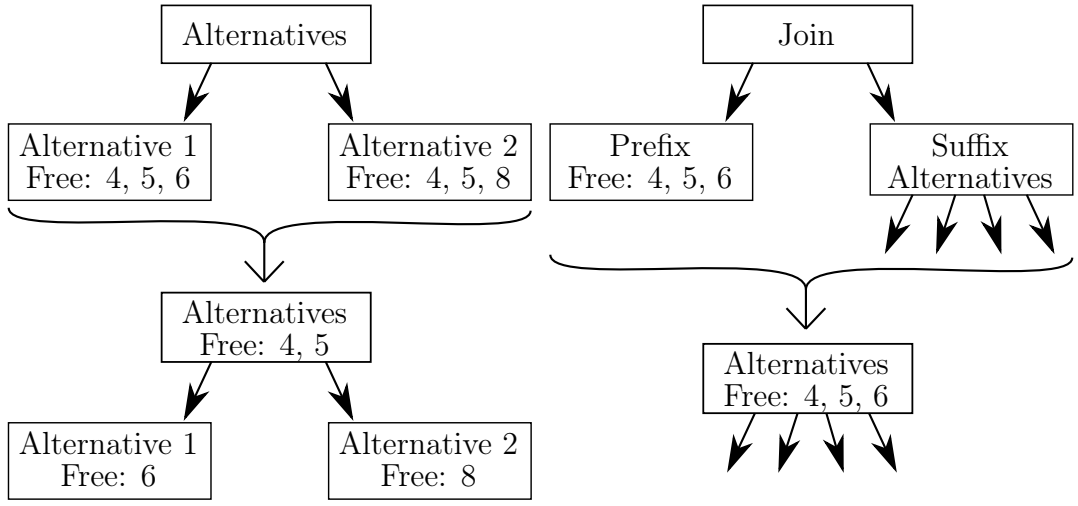


Figure 14: Example of solution tree structure with four foldings

The solution tree is initially constructed as an exact representation of how the backtrack was done. However, this makes the tree significantly larger than necessary. To make the tree smaller and also easier for a human to read, some simplifications can be performed. The Figure 15a shows an alternatives node that has the same two free bases 4 and 5 in each alternative. In such case those free pairs and any common base pairs can be moved into the parent alternatives node. The Figure 15b shows a situation where a join node has a child that is neither a join nor an alternatives node. In such case, that join node and both of its children can be combined into a single node.

The simplified solution tree may contain a large number of base pairs and free bases at its root node. Each of these is necessarily a safe pair or a safe free base: they appear in all foldings represented by the tree. While these lists contain safe base pairs and safe free bases, there is no guarantee that these lists are complete. There may well be other pairs or free bases that are safe, but which could not be brought into the root node using these simplification rules.

A tree walk is used to extract the individual foldings from the solution tree. This walk resembles a depth-first search. At each visited node, the base pairs and free bases of that node are added to a folding. Whenever an alternatives node is encountered, the folding found this far is replicated for each alternative. Join nodes are a more complex case as all possible foldings for both the prefix and the suffix must be evaluated first. The join node then outputs a set of foldings where each prefix



(a) Lifting common parts of the alternatives to the parent node

(b) Combining three nodes into one, when one of them is a leaf node

Figure 15: Simplification of a solution tree

is combined with each suffix. Combining two foldings results in a new folding that has all base pairs and free bases of either source folding.

As the full folding set may be too large to keep in memory, an online algorithm for generating the foldings may be preferable. The tree walk is fairly straightforward to turn into an online algorithm, with the exception of the join nodes. Join nodes would require storing the full folding set from at least one of the prefix and the suffix, or re-evaluating one of the branches many times over. Storing all foldings from a branch may require prohibitive amounts of memory. Re-evaluating one of the branches for each partial folding generated from the other is a better alternative, as it only requires a constant amount of execution time overhead per generated folding.

The solution tree structure is not completely optimal, as it may contain multiple copies of the same subtree. Indeed, whenever the backtrack algorithm examines a specific subsequence (i, j), it produces the same subtree structure. As a sequence of length n contains only $n(n+1)/2$ subsequences, an optimal structure would not contain more nodes than this. This structure, where multiple subsequences may use same shorter subsequence, suggests that the optimal structure might be a directed acyclic graph.

4.11 Degenerate cases during backtracking

Depending on the recurrence relation used, multiple ways to generate the exact same folding may exist. When such degenerate cases exist, it is necessary to take this into account in order to produce the correct folding counts. Different rule sets for maximum pairs algorithms are examined in Section 3.5.

The algorithm descriptions in this thesis are based on the recurrence relations from

the Nussinov algorithm [NJ80] which do not produce degenerate cases. The algorithm implementation is, however, based on the recurrence relation presented in [Sun10] and degenerate cases do appear.

In the rule set used in the algorithm implementation it is necessary to take into account the existence of degenerate cases. Suppose we have range (i, j), $i < j$, that is a loop in the generated secondary structure, i.e. none of the bases in that range are paired. The algorithm can construct this partial answer in $j - i - 1$ different ways, as each split $(i, k), (k + 1, j)$ with k ranging from i to $j - 1$ would produce the same score and the same secondary structure.

Pairing two bases cannot create degenerate cases, since outer pairs are always created before any pairs within them, which creates an ordering for their evaluation. Combining two subsolutions is therefore the only case where degeneracy will need to be considered. Intuitively, if it was possible to check the resulting pairings for each combination that produces optimal score, it would be possible to backtrack only once per unique solution. However, such data is not actually available to the algorithm.

It is possible to leverage the property that pairing cannot cause degeneracy. If the joining case in the backtrack algorithm or the counting algorithm is limited to joining parts (i, k) and $(k + 1, j)$ such that $k + 1$ and j form a base pair or $k + 1 = j$, we are left with only one path to each unique solution. These rules impose an ordering for the decisions, namely one where moving from longer subsequences towards shorter ones splits the sequence into as long as possible prefix and as short as possible suffix.

When joining (i, k) and $(k + 1, j)$, where $k + 1$ and j can be paired, the algorithms must follow only such paths where those two are actually paired. The optimal folding of subsequence $(k + 1, j)$ does not always pair $k + 1$ and j. If the algorithm were to follow all possible ways to generate the subsequence $(k + 1, j)$, many of the degenerate cases would remain.

5 Implementation of the safe and complete algorithm

The safe and complete algorithm is implemented in the Go language. As this is an experimental implementation, it benefits from using a somewhat high-level language with automatic garbage collection, array bounds checking, etc. as opposed to, for example, C++. This is also a somewhat large project, so it benefits from a language that provides good compile-time checks. This rules out, for example, Python which is a duck-typed and run-time checked language. Go also gives good run time performance, which is an interesting property here, as the program needs to process large amounts of data.

When the Go runtime environment is installed, the implementation is available for download through the `go get` tool.

```
go get keltainen.duckdns.org/rnafolding
```

The above `go get` command fetches the source code from a backing Git repository, located at https://keltainen.duckdns.org/git/rnafolding/. Note that at the time of writing this server provides no web UI and this repository can only be used with the `git` tool. The code is also available through the GitHub web UI at `https://github.com/kiirala/rna-safe-complete` and the associated repository. This thesis is written using the Git revision `43fe39b080f553b4e75fe4f5da1a0b2c14efde7e`, committed on 23rd November 2018.

Multiple programs and modes of operation are provided:

`rnafolding` Experimental program that runs multiple different algorithms and checks their results against each other and against sanity checks

- Can read a single strand from FASTA file or large number of strands from tRNA database file [SSHS96] or from dot-bracket files downloaded from STRAND database [ABHC08]
- Analyzes either single sequence outputting a large amount of information to console, or a number of sequences outputting statistics about them
- Can output the folding details of each analyzed sequence into JSON file

`comparesafety` Fast and memory-efficient program for computing safety, built to compare the efficiency of trivial safety algorithm and the dynamic programming version

`trivialsafety` Stand-alone implementation of the trivial safety algorithm. This is intended to read the output of the `RNAsubopt` program from the ViennaRNA package [LBH+11]. It computes the number of solutions and the number of times each base pair and free base appears in those solutions

`dbtofasta` Convert Sprinzl tRNA database [SSHS96] file into set of FASTA files

`fastadump` Dump the internal representation of a sequence read from a FASTA file into a JSON file

5.1 File formats

The FASTA input file format is a simple format that, at its core, consists of a description line and one or more lines containing the bases in ASCII format. Additionally, the folding in dot-bracket format may be appended. An example of this format was given in Figure 9 on page 14. While this format is simple to generate, there exists no definite specification how it should be done. As a result, each system that generates FASTA files does so in a slightly different way and it is complicated to create a system that can read all of them.

As an example of the variations between the different FASTA implementations, the RNA STRAND database [ABHC08] starts comments with the hash character `#` and divides the bases into 50-character lines. The ViennaRNA package [LBH+11], on the other hand, starts comments with greater-than sign `>` and writes the bases on a single line. As a result, attempting to directly process a file from RNA STRAND with ViennaRNA package leads to ViennaRNA package trying to create a folding for each 50-base line separately.

There is also trouble with modified bases. The standard bases are represented by characters `AGCU` and IUB/IUPAC nucleic acid codes [CB85] represent uncertain or unknown bases. There is, however, no clear standard on how non-standard or modified bases should be represented. The documentation for tRNA database [SSHS96] contains a good list of single-character codes for a large number of modified bases, but there is no guarantee that any other system would use the same codes.

In addition to FASTA files, reading text-format tRNA database files [SSHS96] is implemented. This database contains a large amount of data in a more structured format than FASTA files. For the purposes of this thesis, the interesting parts are that this format tells which standard regions the bases belong to, and it provides a reference folding. A tRNA molecule typically has a standard shape, which is shown in Figure 5 on page 6, and knowing which part of this shape each base belongs to allows us to analyze how good the predictions are for each region. An unusual feature of this format is that the foldings are not given in dot-bracket format, but with a simple paired / non-paired indicator. This indicator must be used with knowledge of the standard tRNA folding to determine the actual folding.

The human-readable output from `rnafolding` program represents foldings with ASCII-art visualizations. An example of such representation was seen in Figure 10a on page 15. When safety is included in the visualization, safe bases are rendered with uppercase characters and non-safe bases with lowercase. While this visualization is useful for showing the general molecule structure in simple cases, it is of a limited use. For example, with some molecules, two parts of a molecule will overlap each other, making the visualization impossible to read properly.

When the flag `-outdir` is specified, the `rnafolding` program outputs detailed folding information as JSON files. The format of these output files is described in Appendix 1.

5.2 Sanity checks

To verify that the implementation works correctly, multiple sanity checks are implemented. A sanity check looks at a property of the generated output that should always be true and reports an error if it is not.

A folding is presented internally as array of integers F. In this array $F[i] = j$ when base i is paired with base j. When base i is not paired, $F[i]$ is set to a negative value, as such value clearly represents that the base is unpaired. In this representation, for

any i such that $F[i] \geq 0$, when $F[i] = j$ then also $F[j] = i$ and $i \neq j$. Checking that this holds allows detecting a large variety of issues with the backtrack algorithm: marking only one end of a pair as paired, pairing same base multiple times and writing some non-index values to a folding. Additionally, each pair of a folding can be checked against the original sequence, to see that it is actually possible to pair the two bases.

There are three algorithms implemented. They all produce the maximum number of pairs as a result of the dynamic programming table fill step, and create the foldings separately. All three algorithms should agree on the optimal number of pairs, and each produced folding should have exactly that number of pairs.

Both the Wuchty algorithm and the safe and complete algorithm produce all optimal foldings, so the produced foldings sets should match. Only the order of the individual foldings within the folding set is allowed to be different. The Zuker algorithm, on the other hand, produces only a sample of all optimal foldings, so its output should be a subset of the output of Wuchty algorithm or safe and complete algorithm. A folding set produced by any of the three algorithms must contain each folding only once.

5.3 Limiting solutions

The maximum pairs algorithm typically produces large amounts of foldings. Some heuristics to limit the created outputs are useful, to rule out foldings that are unlikely to occur in nature.

An actual RNA strand can not make very sharp turns. From this stems one good limit, where hairpin loops are required to have a minimum of three bases [WFHS99]. This implementation uses the same limit by default. The minimum loop size can be changed with command line parameter `-minhairpin`.

5.4 Handling modified bases

The tRNA molecules contain a large amount of modified bases, and this is reflected in the tRNA database [SSHS96]. As this folding algorithm only handles the four standard bases, a method for handling the modified bases is needed.

The mapping used in the algorithm implementation is shown in Table 1. The standard bases `CGAU` are kept as-is. Any modified base not present in this list is mapped to the non-pairing base `N`. This mapping matches what is used in Wuchty algorithm [WFHS99] and is based on research specifically made of tRNA properties [Hig95].

Modified base	Symbol	Maps to
Unknown modified adenosine	H	A
2′-O-ribosyladenosine (phosphat)	^	A
Unknown modified cytidine	<	C
2′-O-methylcytidine	B	C
N^4-acetylcytidine	M	C
5-Methylcytidine	?	C
Unknown modified guanosine	;	G
N^2-methylguanosine	L	G
2′-O-methylguanosine	#	G
N^2,N^2-dimethylguanosine	R	G
Unknown modified uridine	N	U
2′-O-methyluridine	J	U
Pseudouridine	P	U
1-Methylpseudouridine	]	U
2′-O-methylpseudouridine	Z	U

Table 1: Mapping from modified bases to standard ones

6 Experimental evaluation

The source data for this evaluation comes from two RNA sequence databases: the Sprinzl tRNA database [SSHS96] and the RNA STRAND database [ABHC08]. The STRAND database is a collection of multiple databases, including the Sprinzl database. Whenever tRNA samples from STRAND database are used, they also include the tRNA samples of the Sprinzl database.

The STRAND database occasionally contains the exact same sequence under multiple database entries. In such cases only one of the identical entries is included. The database also contains sequence fragments. They are not included in the analysis, only full sequences are.

The number of sequences from each database that are used in the following analysis are shown in Table 2. As some tests were too time-consuming when run for 23s rRNA samples, two versions of that dataset are shown. The starred (23s rRNA*) version contains only those sequences where the relevant tests did not time out.

Many of the measurements are made for *decisions*. A decision is either pairing two bases or leaving a base unpaired. For example, a folding presented in dot-bracket notation as $\cdot\cdot((\cdot\cdot))$ contains a total of six decisions: four non-paired bases and two base pairs. Counting decisions means that both base pairs and unpaired bases have an equal weight. Counting metrics per base would mean that a pair has twice as much weight as leaving a base unpaired. When talking of maximum pairing algorithms, all optimal foldings contain the same number of decisions. With minimum free energy algorithm the optimal foldings may have varying number of pairs, so different foldings may contain different number of decisions.

	Number of sequences	Number of bases			
		5th perc.	Average	Median	95th perc.
Sprinzl tRNA	539	72	78	76	87
STRAND tRNA	633	67	78	76	88
STRAND 5s rRNA	136	117	118	120	129
STRAND 16s rRNA	647	954	1536	1531	1869
STRAND 23s rRNA	115	1215	2726	2896	3522
STRAND 23s rRNA*	58	1033	2439	2892	2997

Table 2: Properties of the different datasets used

The safe and complete algorithm gives us the number of times each possible decision appears in the full set of optimal foldings. From these, we can create the *decision fraction*. For example, if there are ten optimal foldings in total and six of them contain the pair (i, j), the decision fraction of that pair is $6/10$ or 60%. This fraction could be interpreted as a probability, but it does not seem to be a true probability. There is no clear indication that the foldings that have been found are evenly distributed and represent an equal probability each.

Mostly these evaluations are made using the safe and complete maximum pairs algorithm implemented for this thesis, specifically the `rnafolding` program described in Section 5. We have also used the *RNAsubopt* program from the *ViennaRNA 2.4.9* package [LBH+11]. This program generates foldings with minimum free energy, which is a better model of RNA folding than maximizing number of base pairs. This program uses the Wuchty algorithm [WFHS99] to generate all possible foldings within a given energy interval from an optimal solution. We have implemented a program named `trivialsafety` that reads the near-optimal foldings from RNAsubopt and uses the trivial algorithm for computing safety from Section 4.5 to compute the same pairing count arrays that a full safe and complete algorithm would have. With this, we can see how well the safe and complete algorithm would perform if it was used with a minimal free energy model, without having to implement the full algorithm beforehand.

Running RNAsubopt on some 16s rRNA and 23s rRNA sequences results in a huge number of near-optimal foldings. To limit the resources needed, tests for 23s rRNA were run with 3-hour CPU time limit first, and an arbitrary set of the remaining ones were run without an explicit time limit. For this reason, we have measurements from RNAsubopt program for only half of the 23s rRNA sequences. These sequences are shown as the starred 23s rRNA* dataset in Table 2. This table shows that these sequences are shorter on average than the 23s rRNA dataset on the whole.

6.1 Number of foldings

Looking at tRNA samples [SSHS96], a maximum pairs algorithm produces a highly variable number of foldings. Sample RY9990, cytoplasmic tRNA from human pla-

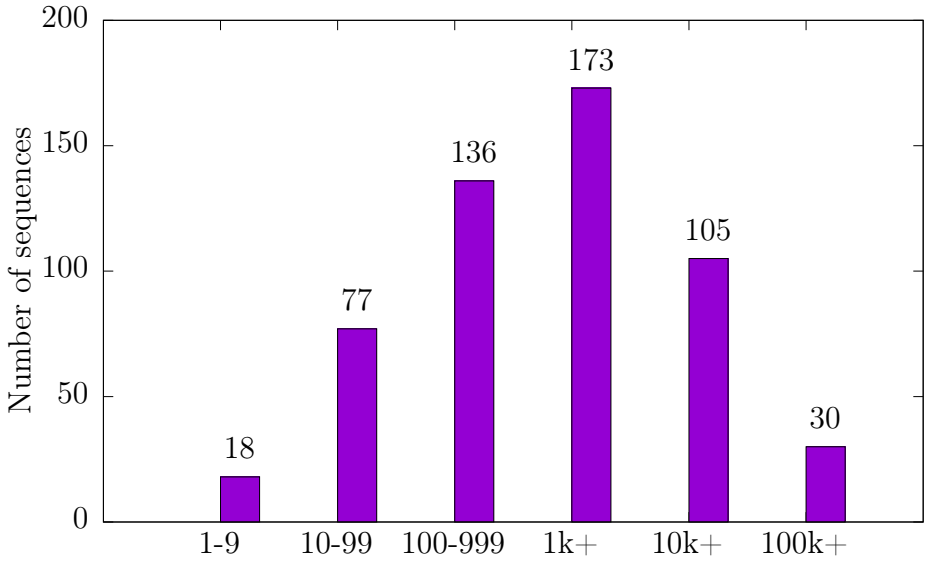


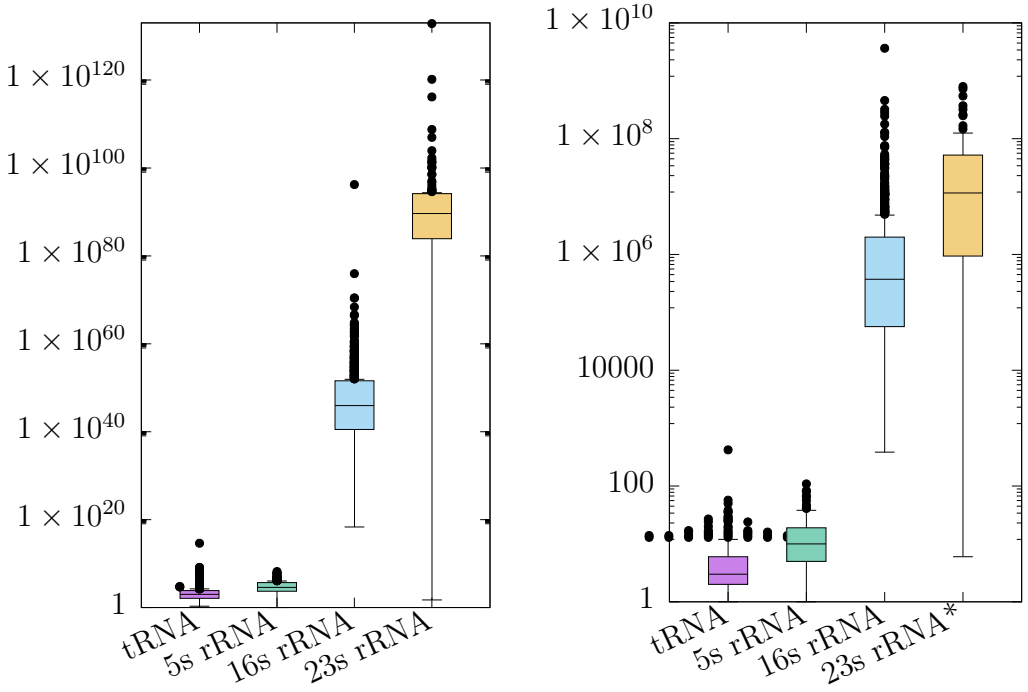
Figure 16: Count of tRNA sequences that result in a given number of possible foldings

centa, has only two optimal foldings with 27 pairs each. On the other hand RX4400, mitochondrial tRNA of green bean, has 505182 optimal foldings with 25 pairs each. The median number of foldings is 1725 over all tRNA samples. Figure 16 shows the distribution of the number of foldings.

Looking into ribosomal RNA in addition to transfer RNA, it becomes soon clear that the maximum pairs algorithm may produce huge amounts of foldings. Figure 17a shows the distribution of how many optimal maximum pairs foldings there are. Figure 17b shows the same plot for the output of RNAsubopt program from ViennaRNA package [LBH+11]. As this program uses the minimum free energy model, the numbers themselves are vastly different compared to the maximum pairs model, but they both give similar results overall.

The RNAsubopt folding count dataset in Figure 17b is computed with a strict suboptimality limit of $\Delta E \leq 1\,\mathrm{kcal/mol}$ in order to save on computation time. Even with this limit, many of the 23s rRNA samples were too complex to evaluate within three hours of allotted time. This certainly means that this figure underestimates the number of foldings RNAsubopt produces for 23s rRNA samples.

Both plots in Figure 17 show that the longer RNA sequences can result in massive numbers of optimal or near-optimal foldings. Evaluating 10^{10} foldings is borderline feasible, taking a couple days worth of CPU time. It would be totally infeasible to evaluate all maximum pairs foldings for 16s rRNA with median of 9×10^{45} foldings per sequence or 23s rRNA with median of 4×10^{89} foldings.



(a) Maximum pairs foldings from the safe and complete algorithm

(b) MFE foldings from ViennaRNA RNAsubopt with $\Delta E \leq 1\,\mathrm{kcal/mol}$

Figure 17: Number of foldings for different types of RNA from STRAND database

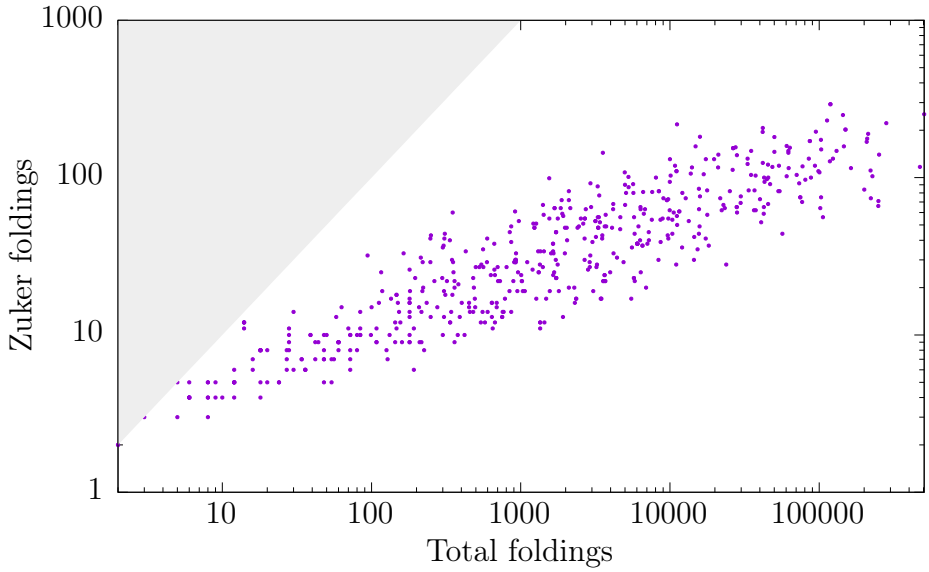


Figure 18: Correlation between total number of optimal foldings and foldings found with Zuker method, using maximum pairs model and sequences from Sprinzl tRNA database [SSHS96]. The gray area is impossible: there the Zuker method would have found more foldings than actually exist

6.2 Correlations between Zuker method and all solutions

The Zuker method [Zuk89] is a common approach to computing multiple optimal foldings. In it, each pair of bases is combined together and a new optimal folding is computed. This method is used in the *mfold* package [Zuk03] and is an available option in the *RNAsubopt* program in the *ViennaRNA* package [LBH+11].

For a sequence of n bases, the forced pair method can find at most $(n \times (n-1))/2$ different foldings, one for each pair of bases. In practice, the algorithm finds a lot less foldings. For example, tRNA sequences [SSHS96] are up to 93 bases long. For such a sequence, the forced pair method could find up to 4278 foldings, which is somewhat higher than the median number of foldings as seen in Section 6.1 but a lot less than the maximum number. In practice, this method finds up to 293 foldings within the tRNA sequences.

As seen in Figure 18, the Zuker method finds a large number of foldings whenever a large number of optimal foldings exists. However, for the same number of total foldings, there is a large spread in the number of Zuker foldings. Taking two examples from Sprinzl tRNA database [SSHS96], the Zuker method finds 72 out of 2098 foldings for tRNA sequence RX7860, but for RG0501 it finds only 16 out of 2304 foldings. The grayed-out area in this figure is impossible to reach, since there the Zuker method would have found more optimal foldings than actually exist. A few points are seen straddling the line in the 1–10 foldings area. These are cases where the Zuker method manages to find all of the optimal solutions.

6.3 Time and memory needed for evaluating solutions

As was seen in Section 4.6, any algorithm that evaluates all solutions individually needs exponential time. The safe and complete algorithm, on the other hand, runs in polynomial time. This is also clearly visible in the experimental results.

The RNAsubopt program from ViennaRNA package [LBH+11] generates all near-optimal MFE foldings. Due to the high number of foldings for larger RNA types, like 23s rRNA, the time needed for evaluating them soon becomes infeasibly large. Relationship between number of generated foldings and the time taken to generate them is shown in Figure 19. Evaluating tRNA and 5s rRNA samples is so fast that measurement noise becomes a significant factor in the measured time, and so they are not shown here.

The Figure 19 also includes a linear fit to the data points. Due to the exponential nature of the data, the value at $x = 0$ is largely irrelevant, and is set to 2 to match the lower end of evaluation times. The resulting least squares fit is $y = 125 \times 10^{-6} \times x + 2$, suggesting that the RNAsubopt program can evaluate approximately $(125 \times 10^{-6})^{-1} = 8000$ foldings per second.

The Figure 20 shows time taken to run the safe and complete algorithm for 16s rRNA and 23s rRNA samples. Comparing this figure with the RNAsubopt timings in Figure 19, it is clear that the safe and complete algorithm does not need to evaluate all solutions individually. For RNAsubopt, the time depends heavily on the number of foldings created, whereas for the safe and complete algorithm such strong effect does not appear. None of the evaluated RNA samples require more than 1000 seconds with the safe and complete algorithm, even though they may generate well over 10^{100} distinct foldings. In comparison, thousand seconds is enough time for RNAsubopt to generate up to 10^7 distinct foldings.

While evaluating all optimal maximum pairs solutions of 16s rRNA and 23s rRNA molecules is completely impractical, tRNA samples are small enough that the required time can be compared. Figure 21 shows the time taken to evaluate different amounts of foldings. The Zuker algorithm stands out clearly here, since it does not actually evaluate all solutions, but a subset of them. When there are only a few optimal foldings, the Zuker algorithm can be the slowest of the three, since it does evaluate all possible base pairs in any case. Between the Wuchty algorithm and the safe and complete algorithm, both of which do evaluate all solutions, the safe and complete seems clearly faster. However, this may well be only a constant factor difference and the implementation of Wuchty algorithm could be optimized to be as fast as the safe and complete algorithm. The Wuchty algorithm needs to make a large number of state copies, which is likely the reason for the lower performance. A promising approach to close this performance gap would be to use copy-on-write storage for the algorithm state representation.

While the compared algorithms in some instances take a huge amount of time, their memory usage is rarely an issue. Table 3 shows statistics of the time and memory usage of RNAsubopt program from ViennaRNA package [LBH+11] combined with

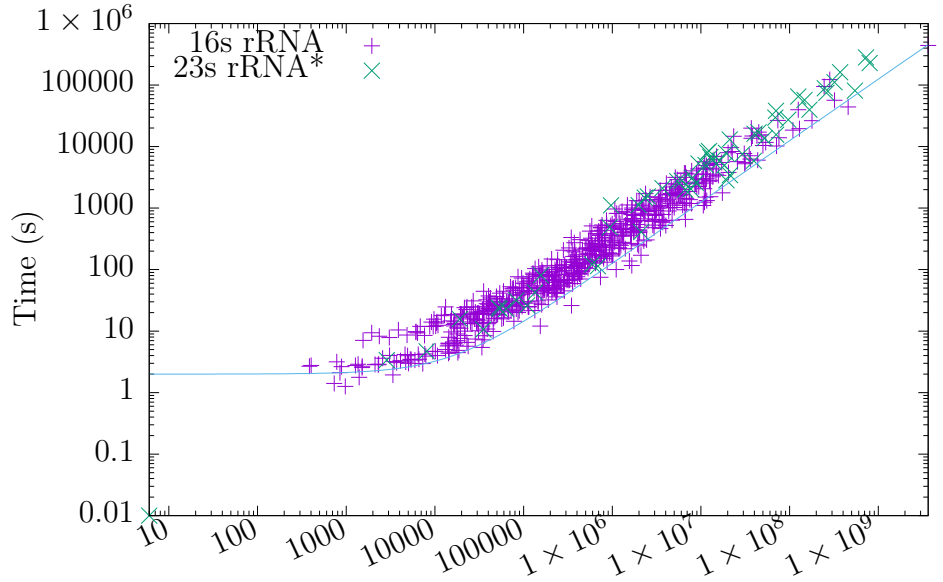


Figure 19: Number of RNAsubopt generated foldings and time taken to generate them

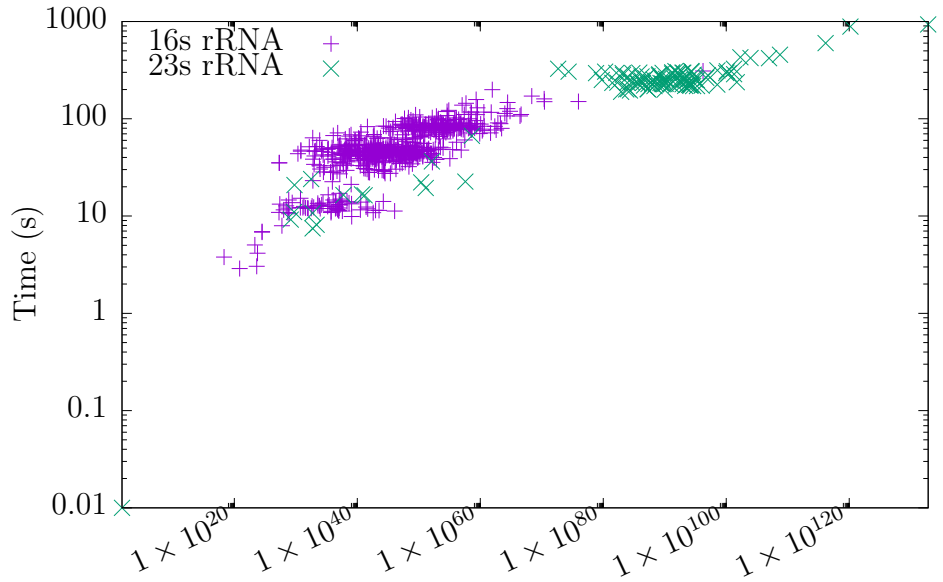


Figure 20: Number of maximum pairs foldings and time taken to evaluate their safety

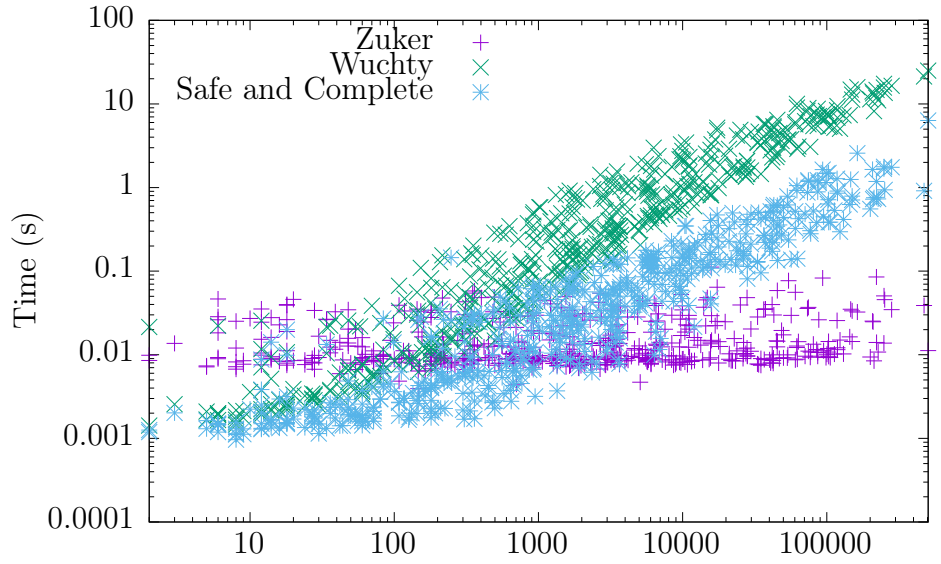


Figure 21: Time taken by the three implemented maximum pairs algorithms to evaluate all foldings

	Time				Memory (MiB)			
	5th perc.	Median	Average	95th perc.	5th perc.	Median	Average	95th perc.
RNAsubopt + `trivialsafety`								
16s rRNA	5 s	136 s	53 min	114 min	91	181	210	322
23s rRNA*	10 s	85 min	9 h	42 h	163	240	293	470
Safe and complete (`comparesafety`)								
16s rRNA	12 s	48 s	56 s	101 s	228	562	597	880
23s rRNA	15 s	4 min	4 min	7 min	353	2233	2051	3070

Table 3: Time and memory usage for computing safety. Comparison of RNAsubopt combined with `trivialsafety` program and of the efficient safe and complete algorithm

the trivial safety algorithm implemented in the `trivialsafety` program and of the efficient safe and complete algorithm implemented in the `comparesafety` program. The memory usage of the RNAsubopt program stays in hundreds of megabytes even in the hardest cases we evaluated. The efficient safe and complete algorithm needs more memory, likely due to the multiple dynamic programming tables containing exact folding counts that have in order of 100 digits. Even then, the memory usage is in a few gigabytes, well within the capabilities of a modest computer.

6.4 Fraction of safe decisions within a sequence

We have evaluated the fraction of safe decisions out of all decisions contained in the optimal folding for a sequence. There are wide variations in this number between

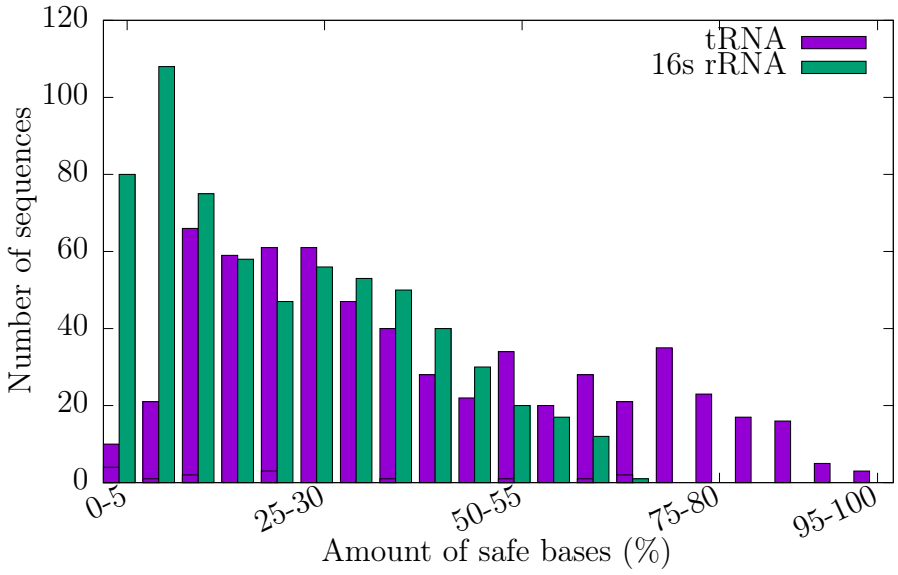


Figure 22: Percentage of safe bases when using a maximum pairs model

different models of RNA folding and different types of RNA.

Figure 22 shows how many sequences there are having a certain fraction of safe decisions when the foldings are predicted with a maximum pairs algorithm. From this figure it is clear that a maximum pairs algorithm tends to produce a rather low number of safe decisions. With tRNA sequences, the median number of safe decisions is 33% and with 16s rRNA it is only 20%. That said, the tRNA sequences also have a fair amount of sequences that have well over 50% of safe decisions, but the longer 16s rRNA sequences lack that long tail. Less than 10% of 16s rRNA samples have over 50% safe decisions.

The MFE model fares better than maximum pairs model when looking at the number of safe decisions. The distribution of sequences is shown in Figure 23. The median fraction of safe decisions is 87% for tRNA and 59% for 16s rRNA. This difference may be in part because the MFE model allowing suboptimal solutions within $\Delta E \leq 1\,\mathrm{kcal/mol}$ from optimal, as used here, produces a lot less foldings than the maximum pairs model. In fact, a fair number of tRNA foldings have 100% safe decisions, meaning there exists only single folding within this energy range.

6.5 Safety and correctness of different tRNA regions

When looking at different distinct parts of tRNA sequences [SSHS96] using the maximum pairs model, some large swings in safety and correctness can be seen. For a reference to different functional parts of tRNA molecule, refer back to Figure 5 on page 6. The fraction of safe predictions and correct predictions per region is shown in Figure 24. The acceptor stem shows a fairly average level of safe bases, but the predicted foldings are quite often correct. The anticodon domain is fairly average

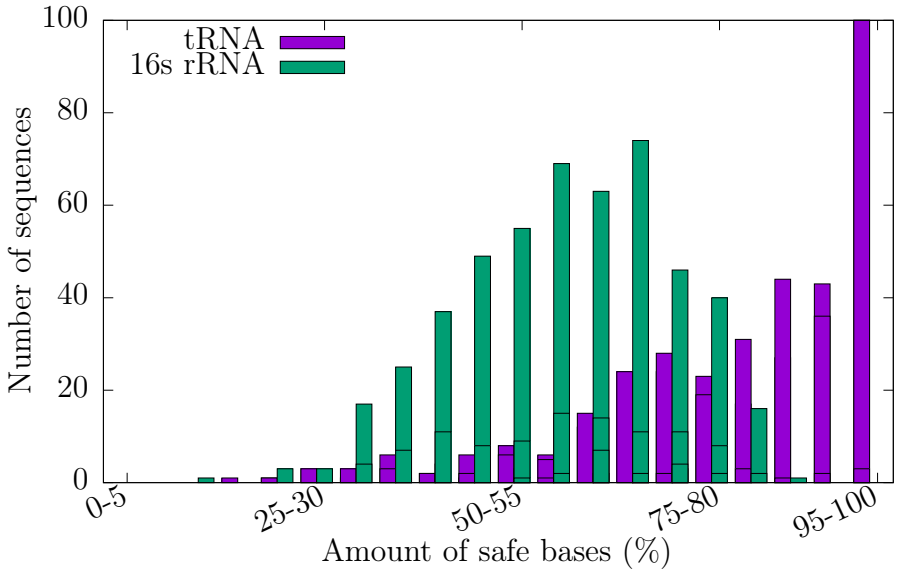


Figure 23: Percentage of safe bases when using an MFE model

	Safe	Non-safe
Free base	45.0 %	34.3 %
Base pair	43.1 %	28.8 %

Table 4: Fraction of safe and non-safe predictions that match biological folding of tRNA sequences using maximum pairs model

in both safety and correctness except for the anticodon itself and some surrounding bases, which are predicted correctly quite often and the predictions are safe. The variable region shows wild swings, but these may be caused by the fact that it is fairly rare to have a variable region of any significant size. The parts on both sides of the centre of the variable region that show especially low safety and correctness appear in no more than 5% of sequences.

Some of the swings in safety and correctness seen in Figure 24 are likely due to how different modified bases are mapped to the standard `GCUA` bases and the non-pairing base `N`. The details of this mapping are discussed in Section 5.4.

Table 4 shows how well the safety derived from a maximum pairs algorithm agrees with the biological folding given in the tRNA database [SSHS96]. In this computation, any bases that map to the non-pairing base `N` are excluded: those never form pairs in this algorithm, so they would always count as safe free bases and obscure how well the algorithm works for actual pair-forming bases.

The prediction probabilities in Table 4 suggest that the safety may work to indicate if a prediction would match reality. It is, however, not a strong indicator, at least when used with a maximum pairs algorithm.

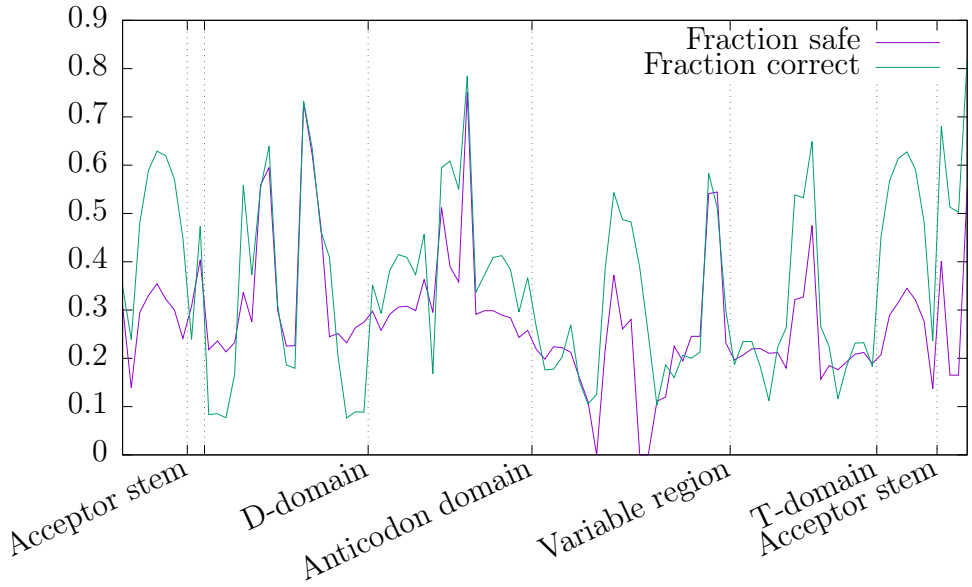


Figure 24: Safety and correctness of tRNA regions

	Safe decisions	Safe	Non-safe
$\Delta E \leq 1.0\,\text{kcal/mol}$	82.4 %	94.5 %	43.3 %
$\Delta E \leq 2.0\,\text{kcal/mol}$	65.9 %	97.3 %	39.6 %
$\Delta E \leq 3.0\,\text{kcal/mol}$	48.5 %	98.1 %	36.7 %
$\Delta E \leq 4.0\,\text{kcal/mol}$	33.5 %	99.0 %	31.1 %
$\Delta E \leq 5.0\,\text{kcal/mol}$	22.8 %	99.6 %	25.6 %

Table 5: Measured probability that a safe or a non-safe prediction of tRNA sequence from MFE algorithm matches the biological folding

6.6 MFE safety at different allowed suboptimality ranges

Table 5 shows how often a safe or a non-safe prediction made by an MFE predictor combined with the trivial safety algorithm matches the biological folding. This data is computed for sequences from Sprinzl tRNA database [SSHS96]. The program RNAsubopt from ViennaRNA package [LBH+11] is used as the MFE predictor and it is combined with the `trivialsafety` program described in Section 5.

The safe parts of a solution are more than twice as likely to be biologically correct than the non-safe parts. This is not a perfect measure of the quality of the predictions, though. If there is a correct non-safe decision concerning base i, there must also exist an incorrect non-safe decision concerning the same base – otherwise the decision would have been safe. Thus it is impossible for non-safe decisions to reach 100% correctness.

Allowing larger ΔE for suboptimal solutions of a minimal free energy algorithm greatly increases the number of foldings found. As we can see from Table 5, the fraction of safe decisions in a folding falls as ΔE is increased and these additional foldings are considered. At the same time, however, those decisions that remain safe

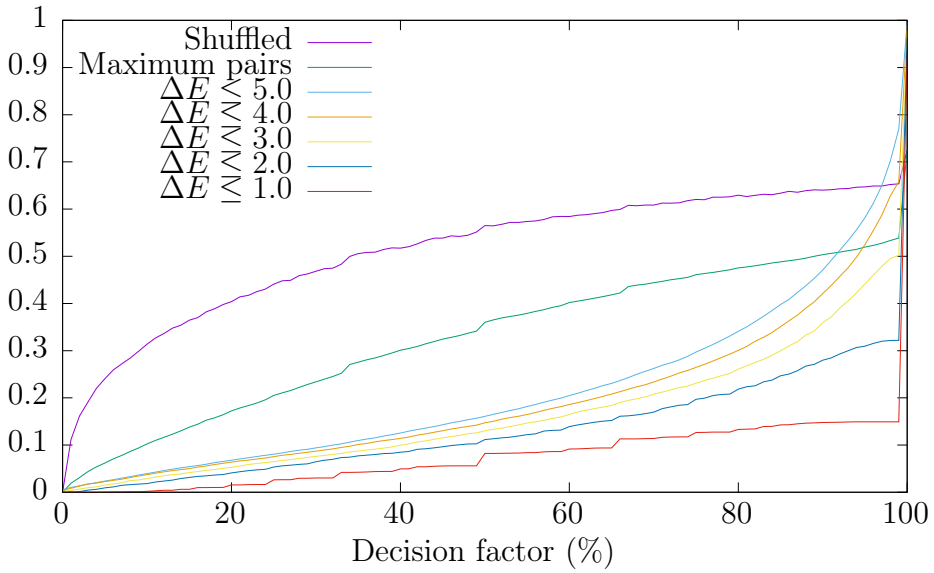


Figure 25: Fraction of biologically correct decisions found at or below a specific decision factor

are increasingly likely to match the biological folding.

Table 5 shows that a safe decision is much more likely to be biologically correct than a non-safe decision. However, it should be noted that a significant number of the non-safe decisions are correct, especially with high values of ΔE. The safety can thus be used to tell which parts of a folding are likely to be correct, but it is not well suited to mark parts as incorrect.

In Figure 25 we can see how the computed decision factors compare between the maximum pairs algorithm and the minimum free energy algorithm. To establish a baseline, the graph also shows a shuffled line: to create this, all predictions with non-zero decision factor made by the maximum pairs algorithm are used but their decision factors are shuffled. This shows how an algorithm with no actual prediction power, but having the same distribution of decision factors, might fare.

Figure 25 shows that the maximum pairs algorithm has a fair amount of prediction power, but it falls greatly short of the minimum free energy algorithm. With the maximum pairs algorithm, the decision factors for biologically correct decisions tend to be all over the range, whereas decision factors given by the minimum free energy algorithm are highly focused to near the 100% mark. It also shows well the effect of increasing number of foldings considered: the folding sets with larger ΔE contain more foldings and tend to give wider range of decision factors than folding sets with low ΔE.

This analysis ignores the fact that a minimum free energy algorithm gives a predicted energy level for each folding. Incorporating these energy levels into the safety algorithm might give us better predictions of whether the base pairs and free bases in the predicted folding match the biological ground truth.

6.7 Precision and recall of safe sections

We have evaluated the precision and recall of safety as predictor of biological correctness. These evaluations compare an arbitrary single folding from an underlying algorithm to the safe parts of the folding. We make these comparisons using both maximum pairs model and MFE model. The maximum pairs value is computed using the `comparesafety` program, as described in Section 5, which implements the efficient safe and complete algorithm. The MFE value is computed using the RNAsubopt program from ViennaRNA package [LBH+11] combined with the `trivialsafety` program that implements the trivial algorithm for computing safety.

In general terms, precision is the fraction of correct predictions out of all predictions, or the probability that any given prediction is correct. Recall is the fraction of correct predictions out of all correct answers, or the probability that any given correct answer appears in the predictions. These general terms can leave room for interpretation, so we will start by defining what these terms mean in our case.

Precision and recall for an arbitrary single folding:

$$\text{precision} = \frac{\text{correct decisions}}{\text{all decisions in output}}$$
$$\text{recall} = \frac{\text{correct decisions}}{\text{decisions in reference}}$$

Precision and recall for safe parts of a set of foldings:

$$\text{precision} = \frac{\text{safe and correct decisions}}{\text{all safe decisions}}$$
$$\text{recall} = \frac{\text{safe and correct decisions}}{\text{decisions in reference}}$$

Looking at the equations for arbitrary single folding, it should be noted that precision and recall are nearly the same. The numerators are exactly the same, and the denumerators nearly so. The number of decisions in the output and in the reference are nearly equal, since the sequence length is the same in both cases and the number of pairs they contain is nearly the same.

The safe parts of a solution have generally better precision than an arbitrary optimal solution. Since using only safe parts of a solution can never introduce new correct predictions, the recall rate of safe parts is necessarily never better than the recall of an arbitrary optimal solution. Figure 26 shows how the safe parts of the foldings predicted with the RNAsubopt program from ViennaRNA package [LBH+11] compare to an arbitrary single folding from the same set.

Table 6 shows the precision and recall rates for an arbitrary optimal solution from the base algorithm and for the safe parts of the solution. Two different algorithms

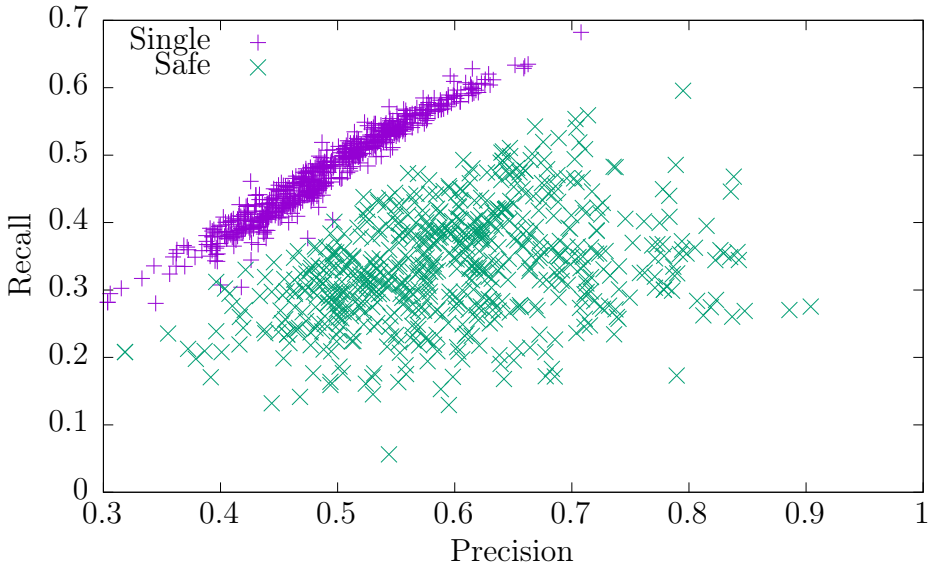


Figure 26: Precision and recall of 16s rRNA samples. Comparison of the first solution from RNAsubopt and of the safe parts.

are used as the base: a maximum pairs algorithm and minimum free energy as implemented in RNAsubopt in ViennaRNA package [LBH+11]. For both precision and recall, the base column shows the precision of the base algorithm. For precision, the safety column shows the precision of the safe parts, and gain column shows the difference of the two in percentage points. For recall, combined column shows the raw recall of the safe parts, which essentially is the combined recall of the base algorithm and of safety used as predictor of correctness. The safety column shows how much of this recall is due to only looking at the safe parts, and is computed as Safety = Combined/Base.

A maximum pairs algorithm is not especially good at predicting foldings, but even then the safe parts are reasonably good predictions of the biological folding. Fig-

		Precision			Recall		
		Base	Safety	Gain	Base	Combined	Safety
Max. pairs	tRNA	45%	67%	23 p.p.	41%	21%	52%
	5s rRNA	36%	53%	17 p.p.	33%	15%	46%
	16s rRNA	27%	42%	15 p.p.	24%	7%	30%
	23s rRNA	29%	43%	14 p.p.	25%	6%	24%
MFE	tRNA	80%	86%	6 p.p.	82%	72%	88%
	5s rRNA	71%	79%	8 p.p.	70%	59%	84%
	16s rRNA	49%	60%	11 p.p.	47%	33%	70%
	23s rRNA*	55%	64%	9 p.p.	52%	38%	72%

Table 6: Average precision and recall of a single prediction compared to the safe parts of the prediction

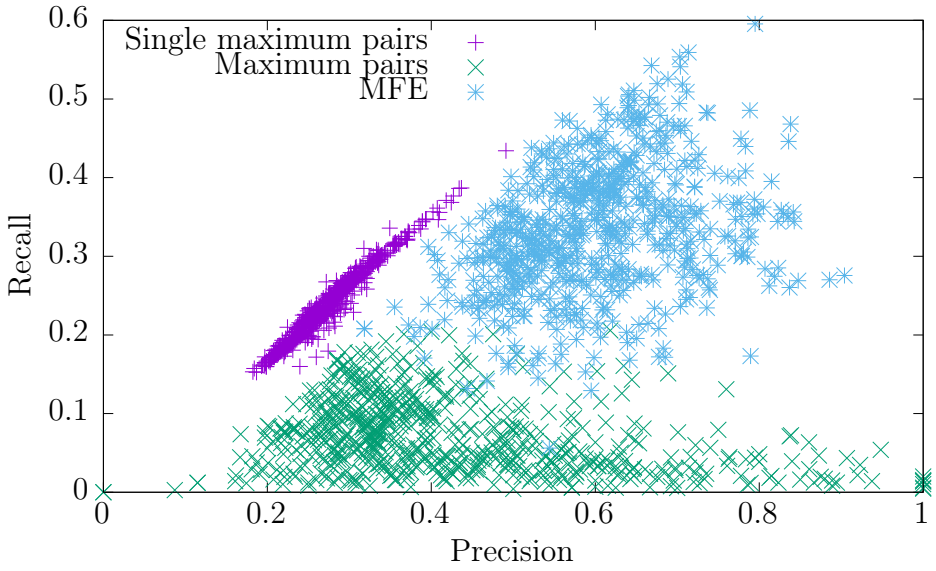


Figure 27: Precision and recall of 16s rRNA samples. Comparison of a single maximum pairs solution and the safe parts of both maximum pairs and minimum free energy foldings.

ure 27 shows how the precision and recall progress when moving from a single maximum pairs folding to the safe parts of a maximum pairs folding and to the safe parts of a minimum free energy folding.

The biological foldings for ribosomal RNA samples typically contain pseudoknots. The precision and recall values presented here suffer since none of the compared algorithms can actually predict pseudoknots.

6.8 Pairing frequency plot

As was seen in Section 3.7, the output of the Zuker algorithm [Zuk89] can be visualized with a dot plot. Such a plot shows all pairings that occur within the optimal foldings that were found with the Zuker method. In a similar fashion, the pairing counts from the safe and complete algorithm can be visualized with a heatmap. An example of such visualization is shown in Figure 28. This figure shows a pronounced diagonal line from top left to bottom right. This line shows how often each base appeared unpaired in the full folding set. Any other points show how often each base pair appeared in the full folding set.

Looking at the example in Figure 28, we can notice some strong bottom left to top right diagonal lines. These indicate stem structures that occur in nearly all optimal foldings. The top right corner shows a similar, but more spread out structure. This indicates that the first seven bases often pair with seven bases near the end, but the exact connections vary. This area corresponds to the tRNA acceptor stem as was seen in Figure 5 on page 6.

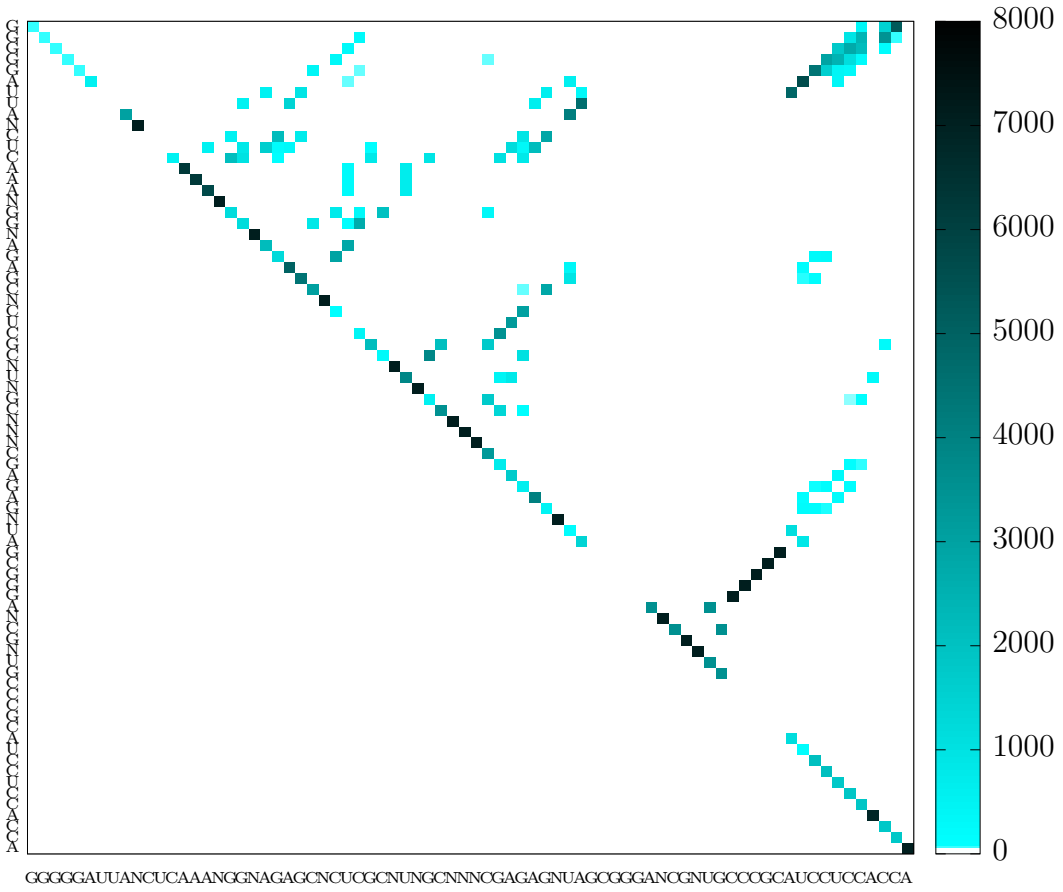


Figure 28: Heatmap of all maximum pairs pairings for human tRNA (RA9990 in Sprinzl database [SSHS96])

7 Conclusions

RNA secondary structure prediction is a helpful tool in several topics, such as in studying evolution of species and in telling apart genes from pseudogenes. The models of RNA folding tend to have a large number of foldings having optimal or near-optimal score. In many cases it is completely infeasible for a human to go through the full set of optimal results, and for larger RNA sequences it becomes problematic to even generate the full set. Hence, methods for handling large result sets are needed and it would be preferable that those methods did not need to evaluate the full result set.

We have presented a safe and complete algorithm for RNA secondary structure prediction. This algorithm can compute how often each base pair or non-paired base occurs in the full set of optimal foldings with no need to evaluate the actual foldings themselves. It is a dynamic programming algorithm that runs in polynomial time in the sequence length. Since there is no need to evaluate the individual foldings, this algorithm can work even with long RNA sequences, such as 23s rRNA that can have in order of 10^{100} foldings with the maximum number of pairs. The algorithm version presented in this thesis predicts maximum pairs foldings. However, it appears possible to extend this algorithm to work with the more accurate minimum free energy model.

In order to verify our approach with real RNA sequences, we have implemented the safe and complete algorithm and, as points of comparison, the Zuker and Wuchty algorithms. All these three algorithm implementations use the maximum pairs model, allowing direct comparisons between the three. Notably, having three algorithms that use different means to achieve the same goal allows cross-verification of the results, which helps to ascertain that the algorithms have been implemented correctly.

The safe and complete algorithm shows good performance when tested with various RNA samples. With longer RNA sequences, such as 23s rRNA, evaluating the full set of solutions is completely impracticial. Even in these cases, the safe and complete algorithm can quickly determine how often each base pair or unpaired base appears in them. Processing the hardest of the evaluated sequences takes a bit over 15 minutes of CPU time and 3 gigabytes of memory.

Experimental results show that both safety and decision fraction are usable as indicators of folding quality. For all types of RNA analyzed, with both the maximum pairs model and the minimum free energy model, the safe parts of a solution are more often correct than the solution as whole. For example, predictions of tRNA folding with MFE model are 80% correct on average while the safe parts are 86% correct, giving 6 percentage point increase in precision. This increase in precision is accompanied by a drop in recall, from 82% to 72%, meaning that the safe parts contain 88% of the correct decisions that were found by the underlying algorithm.

One direction of future work would be to seek other fields where similar algorithm could be applied. Very little of the safe and complete algorithm is actually specific to RNA secondary structure prediction, and the same method based on counting

solutions could be applied to other problems. In order to benefit from the application of this method, the problem should be such that it is typical to have multiple optimal solutions, and that the choice between them has some consequence.

The research into this algorithm would greatly benefit from extending it from the maximum pairs model into a minimum free energy model. While we believe that it is a straightforward extension, an actual implementation would clear any doubt of this. Having such implementation would also help with the experimental results since the output could be directly compared to other state-of-the-art predictor programs.

Another direction of future work would be to investigate if the pairing count table can be used to guide the backtrack algorithm. The current backtracking algorithms produce the optimal foldings in an arbitrary order. Using the pair count table as an input to the backtracking algorithm could help to refine this order. For example, it could be possible to start backtrack from foldings containing common features. Alternatively, it might be possible to derive an uniform sample out of the full folding set.

References

ABHC08 Andronescu, M., Bereg, V., Hoos, H. H. and Condon, A., RNA STRAND: the RNA secondary structure and statistical analysis database. *BMC Bioinformatics*, 9,1(2008), page 340. doi:10.1186/1471-2105-9-340.

Aku00 Akutsu, T., Dynamic programming algorithms for RNA secondary structure prediction with pseudoknots. *Discrete Applied Mathematics*, 104,1(2000), pages 45–62. URL `http://www.sciencedirect.com/science/article/pii/S0166218X00001864`.

ALT91 Antao, V. P., Lai, S. Y. and Tinoco, Jr, I., A thermodynamic study of unusually stable RNA and DNA hairpins. *Nucleic Acids Research*, 19,21(1991), pages 5901–5905.

CB85 Cornish-Bowden, A., Nomenclature for incompletely specified bases in nucleic acid sequences: recommendations 1984. *Nucleic Acids Research*, 13,9(1985), pages 3021–3030.

DLWP04 Dirks, R. M., Lin, M., Winfree, E. and Pierce, N. A., Paradigms for computational nucleic acid design. *Nucleic Acids Research*, 32,4(2004), pages 1392–1403. doi:10.1093/nar/gkh291.

Edd04 Eddy, S. R., How do RNA folding algorithms work? *Nature Biotechnology*, 22,11(2004), pages 1457–1458. doi:10.1038/nbt1104-1457.

GLGW10 Gherghe, C., Leonard, C. W., Gorelick, R. J. and Weeks, K. M., Secondary structure of the mature ex virio Moloney murine leukemia virus genomic

RNA dimerization domain. *Journal of Virology*, 84,2(2010), pages 898–906. URL `https://jvi.asm.org/content/84/2/898`.

Hig95 Higgs, P. G., Thermodynamic properties of transfer RNA: A computational study. *Journal of the Chemical Society, Faraday Transactions*, 91,16(1995), pages 2531–2540. doi:10.1039/FT9959102531.

JP95 Jucker, F. M. and Pardi, A., GNRA tetraloops make a U-turn. *RNA*, 1,2(1995), pages 219–222.

KHH15 Kerpedjiev, P., Hammer, S. and Hofacker, I. L., Forna (force-directed RNA): Simple and effective online RNA secondary structure diagrams. *Bioinformatics*, 31,20(2015), pages 3377–3379. doi:10.1093/bioinformatics/btv372.

LBH$^+$11 Lorenz, R., Bernhart, S. H., Höner zu Siederdissen, C., Tafer, H., Flamm, C., Stadler, P. F. and Hofacker, I. L., ViennaRNA package 2.0. *Algorithms for Molecular Biology*, 6,26(2011). doi:10.1186/1748-7188-6-26.

LC04 Laslett, D. and Canback, B., ARAGORN, a program to detect tRNA genes and tmRNA genes in nucleotide sequences. *Nucleic Acids Research*, 32,1(2004), pages 11–16. doi:10.1093/nar/gkh152.

LE97 Lowe, T. M. and Eddy, S. R., tRNAscan-SE: a program for improved detection of transfer RNA genes in genomic sequence. *Nucleic Acids Research*, 25,5(1997), pages 955–64.

McC90 McCaskill, J. S., The equilibrium partition function and base pair binding probabilities for RNA secondary structure. *Biopolymers*, 29,6-7(1990), pages 1105–1119. doi:10.1002/bip.360290621.

MSZT99 Mathews, D. H., Sabina, J., Zuker, M. and Turner, D. H., Expanded sequence dependence of thermodynamic parameters improves prediction of RNA secondary structure. *Journal of Molecular Biology*, 288,5(1999), pages 911–940. doi:10.1006/jmbi.1999.2700.

MW97 McNaught, A. D. and Wilkinson, A., *Compendium of Chemical Terminology, 2nd ed. (the "Gold Book")*. Blackwell Scientific Publications, Oxford, 1997. doi:10.1351/goldbook. XML on-line corrected version: http://goldbook.iupac.org (2006-) created by Nic, M., Jirat, J. and Kosata, B.; updates compiled by Jenkins, A.

NJ80 Nussinov, R. and Jacobson, A. B., Fast algorithm for predicting the secondary structure of single-stranded RNA. *Proceedings of the National Academy of Sciences of the United States of America*, 77,11(1980), pages 6309–6313. URL `http://www.bibsonomy.org/bibtex/2e695088252ed71e414386cd7c1d8e01c/earthfare`.

RAR97 Rodriguez-Alvarado, G. and Roossinck, M. J., Structural analysis of a necrogenic strain of cucumber mosaic cucumovirus satellite RNA in planta. *Virology*, 236,1(1997), pages 155 – 166. URL http://www.sciencedirect.com/science/article/pii/S0042682297987316.

SB99 Soukup, G. A. and Breaker, R. R., Engineering precision RNA molecular switches. *Proceedings of the National Academy of Sciences*, 96,7(1999), pages 3584–3589. URL http://www.pnas.org/content/96/7/3584.

SC06 Svoboda, P. and Cara, A. D., Hairpin RNA: a secondary structure of primary importance. *Cellular and Molecular Life Sciences*, 63,7(2006), pages 901–908. doi:10.1007/s00018-005-5558-5.

SSHS96 Sprinzl, M., Steegborn, C., Hübel, F. and Steinberg, S., Compilation of tRNA sequences and sequences of tRNA genes. *Nucleic Acids Research*, 24,1(1996), pages 68–72. doi:10.1093/nar/24.1.68.

ST18 Salmela, L. and Tomescu, A. I., Safely filling gaps with partial solutions common to all solutions. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*. doi:10.1109/TCBB.2017.2785831.

Sta99 Stanley, R. P., *Enumerative combinatorics: Vol 2.* Cambridge University Press, 1999.

Sun10 Sung, W.-K., *Algorithms in bioinformatics : a practical introduction*. Boca Raton (Fla): Chapman & Hall/CRC, 2010.

THK+04 Tamura, M., Hendrix, D. K., Klosterman, P. S., Schimmelman, N. R. B., Brenner, S. E. and Holbrook, S. R., SCOR: Structural classification of RNA, version 2.0. *Nucleic Acids Research*, 32,suppl_1(2004), pages D182–D184. doi:10.1093/nar/gkh080.

TM17 Tomescu, A. I. and Medvedev, P., Safe and complete contig assembly through omnitigs. *Journal of Computational Biology*, 24,6(2017), pages 590–602. doi:10.1089/cmb.2016.0141.

WB85 Waterman, M. S. and Byers, T. H., A dynamic programming algorithm to find all solutions in a neighborhood of the optimum. *Mathematical Biosciences*, 77,1(1985), pages 179–188. URL http://www.sciencedirect.com/science/article/pii/0025556485900963.

WFHS99 Wuchty, S., Fontana, W., Hofacker, I. L. and Schuster, P., Complete suboptimal folding of RNA and the stability of secondary structures. *Biopolymers*, 49,2(1999), pages 145–165. doi:10.1002/(SICI)1097-0282(199902)49:2<145::AID-BIP4>3.0.CO;2-G.

WGA+02 Waugh, A., Gendron, P., Altman, R., Brown, J. W., Case, D., Gautheret, D., Harvey, S. C., Leontis, N., Westbrook, J., Westhof, E., Zuker, M. and Major, F., RNAML: A standard syntax for exchanging RNA information. *RNA*, 8,6(2002), pages 707–717. doi:10.1017/S1355838202028017.

WKW90 Woese, C. R., Kandler, O. and Wheelis, M. L., Towards a natural system of organisms: proposal for the domains archaea, bacteria, and eucarya. *Proceedings of the National Academy of Sciences*, 87,12(1990), pages 4576–4579. URL `http://www.pnas.org/content/87/12/4576`.

WTK+94 Walter, A. E., Turner, D. H., Kim, J., Lyttle, M. H., Müller, P., Mathews, D. H. and Zuker, M., Coaxial stacking of helixes enhances binding of oligoribonucleotides and improves predictions of RNA folding. *Proceedings of the National Academy of Sciences*, 91,20(1994), pages 9218–9222. URL `http://www.pnas.org/content/91/20/9218`.

ZMT99 Zuker, M., Mathews, D. H. and Turner, D. H., Algorithms and thermodynamics for RNA secondary structure prediction: a practical guide. In *RNA biochemistry and biotechnology*, Barciszewski, J. and Clark, B., editors, volume 70 of *NATO Science Series (Series 3: High Technology)*, Springer, Dordrecht, 1999, pages 11–43, doi:10.1007/978-94-011-4485-8_2.

Zuk89 Zuker, M., On finding all suboptimal foldings of an RNA molecule. *Science*, 244,4900(1989), page 48–52. doi:10.1126/science.2468181.

Zuk03 Zuker, M., Mfold web server for nucleic acid folding and hybridization prediction. *Nucleic Acids Research*, 31,13(2003), pages 3406–3415. doi:10.1093/nar/gkg595.

Appendix 1. JSON output of `rnafolding` program

The `rnafolding` program outputs detailed folding information as JSON files, when the `-outdir` flag is specified. Each output file contains one JSON object structured as follows.

Name Name of the sequence. For tRNA database and RNA STRAND files, this is the sequence ID in the database. For FASTA files, this is the first word of the sequence description

Comment Full contents of the sequence description

Sequence Sequence bases as ASCII string

SequenceWithSafety Same as Sequence, except with safe bases in upper case characters and non-safe bases in lower case

Rules Rules applied to folding

- **MinHairpin** Minimum number of non-paired bases that must appear inside a loop

Timing Time taken for different parts of the execution

- **ZukerSeconds** Time to compute foldings with the Zuker algorithm
- **WuchtySeconds** Time to compute foldings with the Wuchty algorithm
- **SafeCompleteSeconds** Time to produce the folding tree with the safe and complete algorithm
- **PairArraysSeconds** Time to create the full list of foldings from the folding tree
- **SafeCompleteTotalSeconds** Total time to produce full list of foldings with the safe and complete algorithm. Comparable to ZukerSeconds and WuchtySeconds.
- **TrivialSafetySeconds** Time to compute safety using the trivial algorithm, as in Section 4.5.
- **SafeCompleteSafetySeconds** Time to compute safety using the dynamic programming algorithm, as in Section 4.6.

Counts Numbers of various processed and generated items

- **SequenceBases** Number of bases in the sequence
- **OptimalPairs** Maximum number of pairs for the sequence
- **ZukerFoldings** Number of unique foldings generated by the Zuker algorithm
- **WuchtyFoldings** Number of unique foldings generated by the Wuchty algorithm
- **SafeCompleteFoldings** Number of unique foldings generated by the safe and complete algorithm. Should equal WuchtyFoldings
- **SafeBases** Number of safe bases. Should be $0 \leq SafeBases \leq SequenceBases$

Sanity Contains various strings, which are filled only if some sanity check fails. The strings will contain a human-readable description of what failed

ReferenceFolding Biological folding, as read from the input file

ZukerFoldings Array of all foldings generated with Zuker algorithm

AllFoldings Array of all optimal foldings

SafeCompleteFoldingTree The folding tree generated by the safe and complete algorithm in human-readable text format

Safety Details of safety determined from all optimal foldings

- **SafeBase** Array of booleans, where element i is true if base at position i is safe (part of safe pair or safe unpaired base)
- **PairCount** Matrix of integers, where $PairCount[i][j]$, $i < j$, is the number of optimal foldings that include the pair (i, j)
- **FreeCount** Array of integers, where $FreeCount[i]$ is the number of optimal foldings that leave the base i unpaired

ReferencePosition Array of integers, where $ReferencePosition[i]$ contains the position of base i in the standard tRNA folding. Only applicable for foldings derived from the Sprinzl tRNA database [SSHS96]