Safe and Complete Algorithms for Dynamic Programming Problems, with an Application to RNA Folding

Niko Kiirala
Department of Computer Science and Helsinki Institute for Information Technology HIIT, University of Helsinki, Finland
kiirala@cs.helsinki.fi

Leena Salmela[1]
Department of Computer Science and Helsinki Institute for Information Technology HIIT, University of Helsinki, Finland
leena.salmela@helsinki.fi

Alexandru I. Tomescu[1]
Department of Computer Science and Helsinki Institute for Information Technology HIIT, University of Helsinki, Finland
alexandru.tomescu@helsinki.fi

Abstract

Many bioinformatics problems admit a large number of solutions, with no way of distinguishing the correct one among them. One approach of coping with this issue is to look at the partial solutions common to all solutions. Such partial solutions have been called *safe*, and an algorithm outputting all safe solutions has been called *safe and complete*. In this paper we develop a general technique that automatically provides a safe and complete algorithm to problems solvable by dynamic programming. We illustrate it by applying it to the bioinformatics problem of RNA folding, assuming the simplistic folding model maximizing the number of paired bases. Our safe and complete algorithm has time complexity $O(n^3M(n))$ and space complexity $O(n^3)$ where n is the length of the RNA sequence and $M(n) \in \Omega(n)$ is the time complexity of arithmetic operations on $O(n)$-bit integers. We also implement this algorithm and show that, despite an exponential number of optimal solutions, our algorithm is efficient in practice.

2012 ACM Subject Classification Theory of computation → Dynamic programming; Applied computing → Life and medical sciences; Applied computing → Molecular structural biology; Theory of computation → Design and analysis of algorithms

Keywords and phrases RNA secondary structure, RNA folding, Safe solution, Safe and complete algorithm, Counting problem

Digital Object Identifier 10.4230/LIPIcs.CPM.2019.8

Funding *Leena Salmela*: Supported by Academy of Finland (grants 308030 and 314170).

1 Introduction

Many bioinformatics problems ask to reconstruct a biological object based on some data observed from it. However, in many cases, such data is incomplete or erroneous, and the bioinformatics problem admits many solutions, with no way of distinguishing the correct one among them. One approach is to enumerate all solutions, or only the first k best solutions to the problem. Then, "one can apply more sophisticated quality criteria, wait for data to become available to choose among them, or present them all to human decision-makers" [11].

[1] Equal contribution

For example, this approach was applied to the reconstruction and analysis of metabolic pathways [5, 4] and gene regulation networks [24]. See the surveys [11, 12] for more details on best-k enumeration. However, in many other cases this approach is infeasible because there may be a large number of solutions, even exponential in the input length. In such cases, another approach is to consider only the partial solutions common to all optimal or near-optimal solutions. Due to the large number of solutions, it is often infeasible to enumerate all solutions and then find the parts common to all of them.

In the past this approach has been applied e.g. to sequence alignment. The aligned symbols common to all optimal and near-optimal alignments of two biological sequences were shown to be efficiently retrievable in [29, 28, 9, 14, 36]. In [29] these were called "reliable regions" and were shown to match in a significant proportion the true ones determined experimentally from tertiary structure superpositions.

More recently, this approach was applied to the contig assembly stage of the genome assembly problem. This requires to reconstruct strings as long as possible that are guaranteed to occur in any genome that could have generated the sequenced reads. In [26] such a contig assembly algorithm outputting strings common to all genome assembly solutions was called *safe*. A safe algorithm outputting *all* such safe strings was called *complete*. For the theoretical assembly model where the assembly solution is an Eulerian cycle, a safe and complete algorithm was suggested in [20]. Safe and complete algorithms were also studied for the gap filling problem [22], which is another stage of the genome assembly problem.

1.1 Contribution

In this paper we study another bioinformatics problem which admits a large number of optimal solutions in practice: the RNA folding problem. (We will formally introduce it and discuss it in greater detail in Sec. 1.2.) We will consider a basic model of RNA folding, namely the one maximizing the number of paired RNA bases, solvable by dynamic programming.

As opposed to the previous safe and complete algorithms tailored for each separate problem, we develop here a technique having the advantage that it can produce safe and complete algorithms for many problems solvable by dynamic programming. The only restriction is that the safe partial solutions it can find are required to be identifiable from the dynamic programming recurrence.

Our technique is based on counting solutions and works as follows: (i) for every substring of the input string, it counts the number of optimal solutions for it; (ii) for every substring of the input string, it counts the number of full optimal solutions that also use an optimal solution of the substring; (iii) for every possible partial solution identifiable from the dynamic programming recurrence (i.e., base pair, or unpaired base), it counts in how many full optimal solutions it appears. This base-pair maximization model for RNA folding can be solved in time $O(n^3)$, where n is the length of the RNA molecule. Our safe and complete algorithm for it reports the bases paired in all optimal solutions, and the bases unpaired in all optimal solutions. It has time complexity $O(n^3 M(n))$ and space complexity $O(n^3)$, where $M(n) \in \Omega(n)$ is the time complexity of arithmetic operations on $O(n)$-bit integers. Thus, our algorithm is also efficient, in the sense that its time complexity has no additional overhead with respect to computing a single optimal solution, except for the $M(n)$ factor.[2]

[2] Note that computing an optimal folding under the base-pair maximization model can be sped up to $O(n^3/\log n)$ using the Four Russians technique, see [13]. It would be interesting to study if this technique can be applied to speed up also our algorithm.

Our technique generalizes, and has some similarities with existing algorithms. For example, it generalizes the previous safe and complete algorithm [29] reporting aligned symbols appearing in all optimal alignments. However, our algorithm is strictly more general: in the sequence alignment problem each case of the recurrence depends only on one previously computed value, whereas our technique encompasses problems where several previously computed values are combined. Our algorithm is also similar to the inside-outside algorithm [6] used e.g. in EM estimation of probabilistic context-free grammars. The first step of our algorithm, counting the number of optimal solutions for any input substring, resembles the inside algorithm, whereas the second step is similar to the outside algorithm. Another approach whose aim is also to provide general solutions to problems solvable by dynamic programming is the Algebraic Dynamic Programming approach of Giegerich et al. [15]. This also provides a general algorithm for counting full solutions to dynamic programming algorithms. However, it does not count full solutions containing a given partial solution, and thus does not provide insights about safety.

Other dynamic programming problems where our techniques provides a safe and complete algorithm include, just to name a few: finding the parenthesization of an arithmetic expression [15] maximizing or minimizing its value, finding an optimal order in which to multiply a chain of matrices [15], finding a maximum-weight independent set in a node-weighted tree. In each such case the partial solutions that can be found as safe are the ones that are identifiable from the dynamic programming recurrences. For example, for the first problem, the parentheses common to all optimal solutions, and for the last problem, the nodes common to all optimal solutions. In Sec. 5 we will explain this more formally, in detail.

1.2 RNA folding

RNA is a single-stranded molecule that consists of bases `A`, `C`, `G`, and `U`. RNA folds upon itself so that base `A` pairs with `U` and `C` pairs with `G`. Base `G` can also pair with `U`, forming a wobble pair. The pairing of the bases forms the secondary structure of an RNA molecule. This structure is important for the biological function of many classes of RNA molecules, such as ribosomal RNA (rRNA) and transfer RNA (tRNA). Thus, in order to predict the biological function of an RNA molecule, we first need to predict its secondary structure.

When folding upon itself, an RNA molecule attempts to find a state which is energetically optimal. Traditional RNA secondary structure prediction algorithms, such as the Nussinov algorithm [21], have formulated the problem as finding a secondary structure which maximizes the number of paired bases. Zuker [35] extended this model to minimize free energy, which is biologically more accurate. Later, the free energy model has been refined when the contribution of different substructures has been better understood [30, 18]. However, none of these models is fully accurate or complete. For example, although some dynamic programming algorithms allow a limited form of pseudoknots (see e.g. [2]), many other dynamic programming solutions do not attempt to predict pseudoknots. Additionally, RNA does not always fold in the globally optimal shape, but instead sometimes takes a locally optimal shape, for example in the presence of other molecules.

The solution to the RNA secondary structure prediction problem is very rarely unique. In the maximum-pairs formulation, a large number of optimal solutions can usually be found. The minimum free energy formulation is more fine grained, but still, if one looks at optimal *and* near-optimal solutions, then again their number is large. There are several established ways of coping with the issue of multiple solutions. The algorithm by Wuchty et al. [33] enumerates all optimal and suboptimal solutions, and therefore its output can be too large to utilize. The Zuker algorithm [35] considers each possible pair of bases at

a time, and computes the secondary structure with best score that contains that pair of bases. However, this method might not find all optimal or close to optimal solutions, since the number of solutions can be exponential in the length of the RNA sequence, whereas the maximal number of different base pairs is only quadratic. When assuming the free energy model, McCaskill [19] calculates the equilibrium partition function for an RNA secondary structure. Based on this, the binding probabilities under the free energy model are computed for each pair of bases, over all possible secondary structures, not only the optimal ones. This can be done in cubic time, see [19] or [10]. More recently, Zakov et al. [34] showed how to reduce the time complexity of computing the binding probabilities under the free energy model to sub-cubic time, using ideas from Valiant's sub-cubic algorithm for context free grammar recognition, see [27, 1].

In order to clearly illustrate our technique for obtaining general safe and complete algorithms for dynamic programming problems, in this paper we focus on the simplistic model of RNA folding maximizing the number of paired bases. We implement our safe and complete algorithm for it and run experiments on real RNA molecules to show how it performs in practice.

In the Appendix, we also study experimentally the biological relevance of the notion of safety for the RNA folding problem. For this purpose, we also implement a trivial safe and complete algorithm, which, given all optimal and sub-optimal RNA foldings under the minimum free energy model reported by the ViennaRNA package [17], it finds the base pairs common to all such foldings. We observe that for the maximum pairs formulation, the precision of the safe sub-solutions are improved by up to 72% (median value), respectively, as compared to a single, entire, solution. For the minimum free energy formulation, the increases of precision of the safe sub-solutions are up to 22% (median value).

However, the running time of this naive safe and complete algorithm for the free energy model is prohibitively large for longer RNA sequences, whereas our algorithm for the maximum pairs formulation is efficient in practice. Since also the minimum free energy model is solvable by dynamic programming (using four dynamic programming tables), it is relevant as future work to apply our technique also to this more complex dynamic programming algorithm.

2 Preliminaries

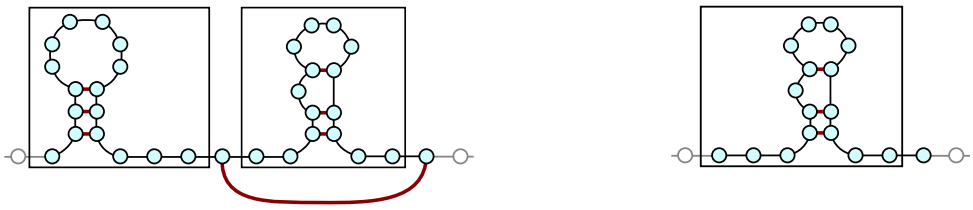
In this section we introduce the basic notation and definitions. We then describe Nussinov's dynamic programming algorithm for RNA folding, which is the starting point of our safe and complete algorithm.

In this paper we assume that $R \in \{\mathtt{A}, \mathtt{C}, \mathtt{G}, \mathtt{U}\}^n$ is a string over the RNA alphabet, indexed from 1 to n. We denote by $R(i, j)$ its substring starting at position i and ending at position j, inclusively ($1 \leq i \leq j \leq n$). The following RNA symbols *can be paired*: $\{\mathtt{A}, \mathtt{U}\}$, $\{\mathtt{C}, \mathtt{G}\}$, $\{\mathtt{G}, \mathtt{U}\}$. A *folding* of R is a set of pairs of indices in R (also called *base pairs*) $\{\{i_1, j_1\}, \{i_2, j_2\}, \ldots, \{i_t, j_t\}\}$ such that:

- $i_1, j_1, i_2, j_2, \ldots, i_t, j_t$ are all distinct, and
- $R(i_k)$ can be paired with $R(j_k)$, for all $k \in \{1, \ldots, t\}$.

A *pseudoknot* in such a folding consists of two pairs of indices, say $\{x, y\}$ and $\{z, w\}$ ($x < y$ and $z < w$), such that z is in the interval (x, y), but w is outside the interval (x, y). We next introduce the main problem tackled in this paper.

▶ **Problem 1** (Maximum pairs formulation). *Given a string $R \in \{\mathtt{A}, \mathtt{C}, \mathtt{G}, \mathtt{U}\}^n$, find a folding of R without pseudoknots maximizing the number of base pairs.*



(a) Pair $R(k)$ and $R(j)$. **(b)** Append $R(j)$ as unpaired.

Figure 1 Cases in Nussinov's algorithm.

A well-known algorithm for solving Problem 1 is Nussinov's algorithm [21]. This is based on dynamic programming, and runs in time $O(n^3)$ and requires $O(n^2)$ space. This algorithm is conceptually simple, and thus a good candidate to be extended into a safe and complete algorithm for Problem 1. Since all algorithms considered in this paper will be about foldings without pseudoknots, from now onwards by "folding" we mean one without pseudoknots.

Nussinov's algorithm defines $V(i,j)$ as the number of base pairs in an optimal folding of the substring $R(i,j)$. The maximum number of base pairs in a folding for R will be $V(1,n)$, and such a folding can be obtained by standard backtracking through the table V.

We initialize $V(i,j) = 0$ for $i \geq j$, since a sequence of length at most one cannot have any paired bases. The values $V(i,j)$ are computed with the following recurrence:

$$V(i,j) = \max \begin{cases} \max \Big\{ V(i,k-1) + V(k+1,j-1) + 1 \mid \\ \qquad i \leq k < j \text{ and } R(k) \text{ can be paired with } R(j) \Big\}, \\ V(i,j-1). \end{cases} \tag{1}$$

The two cases of this recurrence are also shown in Figure 1. In the first case, Nussinov's algorithm attempts to pair any base $R(k)$, where $k \in \{i, \ldots, j-1\}$, with the base $R(j)$. Whenever this pairing is possible, it uses the previously computed answers for the shorter substrings $R(i,k-1)$ and $R(k+1,j-1)$. The second case applies when $R(j)$ is not paired with any base $R(k)$: the algorithm takes the optimal folding of $R(i,j-1)$ and appends the base $R(j)$ to it as non-paired.

Nussinov's algorithm generates only a single optimal folding. However, it can be modified to output all optimal and suboptimal foldings, through a general technique introduced by [31] for the shortest path problem. This technique was applied to the RNA folding problem by Wuchty et al. [33]. In fact, [33] applies this technique to both models for RNA folding, maximum pairs and minimum free energy. The minimum free energy version is included in the RNAsubopt program contained in the ViennaRNA package [17].

3 A safe and complete algorithm for RNA folding

We say that a base pair $\{i,j\}$ is *safe*, if for any optimal folding $F = \{\{i_1,j_1\},\{i_2,j_2\},\ldots,\{i_t,j_t\}\}$ of R, we have that $\{i,j\} \in F$. Likewise, we say that an (unpaired) base i is *safe*, if for any such optimal folding F, we have that i is not paired with any other base. Formally, if for all optimal foldings F and all $\{x,y\} \in F$, we have $i \notin \{x,y\}$.

A safe and complete algorithm for Problem 1 is one outputting all safe base pairs and safe unpaired bases. We start by describing a trivial safe and complete algorithm. In Sections 3.1–3.3 we describe our efficient algorithm.

As mentioned above, Nussinov's algorithm can be extended to generate all optimal foldings. Let us assume that we have generated all of them, and that their number is s. We can then naively go through each optimal folding, and for each of the $O(n^2)$ possible base pairs of R, and each unpaired base, keep track if it appears in all optimal foldings. This trivial algorithm takes overall time $O(sn + n^2)$. The $O(sn)$ term comes from reading through every folding, and the $O(n^2)$ term comes from initializing the base-pairs matrix. Note however that this complexity is not polynomial in n, as there can be RNA strings of length n admitting an exponential number s of foldings (see our experimental results). Also in practice this complexity is prohibitive; for example, even some tRNA samples that are around 80-bases long have hundreds of thousands of solutions.

We will next describe a polynomial-time safe and complete algorithm, based on Nussinov's algorithm. The main idea is to derive, for every base pair, the number of optimal foldings in which it appears; and analogously, for every base, the number of optimal foldings in which it appears as unpaired. If in addition we also know the total number of optimal foldings, then we can decide safety based on checking if these numbers are equal or not.

These numbers will be derived in a sequence of steps, each computing a numeric table by dynamic programming. The table from each step will be used in the computation of the table for the next steps. These steps are:

1. Compute the number of optimal foldings of any substring $R(i, j)$ (Section 3.1)
2. Compute the number of optimal foldings of R, that use an optimal folding of the substring $R(i, j)$ (Section 3.2)
3. Compute the number of optimal foldings of R containing a given base pair, or a given unpaired base. (Section 3.3)

When analyzing the time complexity of this algorithm, we have to take into account that the number of foldings may be exponential in the input size. As such, we cannot assume constant-time integer arithmetic operations.

It is not hard to see that one upper bound on the total number of foldings of any RNA string of length n is 3^n. Any possible folding corresponds to a string of balanced parentheses (the paired bases), in which we have interspersed a third character standing for the non paired bases. The number of such ternary strings of length n is trivially at most 3^n. Such numbers take $O(\log_2(3^n)) = O(n)$ bits to represent. Tighter bounds on the number of optimal RNA foldings under the base-pair maximization model exist, see e.g. [16]. We note that additions on n-bit numbers can be performed in time $O(n)$, and we denote by $M(n) \in \Omega(n)$ the time needed to multiply or divide two $O(n)$ bit numbers.[3] Our safe and complete algorithm for Problem 1 will take $O(n^3 M(n))$ time and $O(n^3)$ space.

3.1 The number of optimal foldings of any subsequence

Given $i \leq j \in \{1, \ldots, n\}$, we let $S(i, j)$ be the number of optimal foldings of $R(i, j)$. The total number of optimal foldings of R will thus be obtained as $S(1, n)$. Computing $S(i, j)$ can be done by dynamic programming, with a recurrence analogous to Nussinov's recurrence (1), and also using Nussinov's table V.

As base case, observe that any sequence containing at most one base clearly has exactly one optimal folding. In other words, $S(i, j) = 1$ whenever $i \geq j$.

[3] For example, with the Schönhage-Strassen method [23], two n-bit numbers can be multiplied in time $M(n) = O(n \log n \log \log n)$.

As in Nussinov's recurrence, there are two ways of producing an optimal folding of $R(i,j)$: by pairing $R(j)$ with some $R(k)$, for some $k \in \{i,\ldots,j-1\}$, or appending $R(j)$ at the end of the optimal folding of $R(i,j-1)$. For each of these cases, and for each value of k, we obtain a distinct folding. As such, to obtain $S(i,j)$ we can sum up the number of optimal foldings in each case.

In the first case, when pairing $R(j)$ with $R(k)$, there are two substrings to consider: $R(i,k-1)$ and $R(k+1,j)$. Every possible combination of optimal foldings of these substrings will produce a unique optimal folding to the larger substring $R(i,j)$. Such combination thus contributes $S(i,k-1) \times S(k+1,j)$ optimal foldings to $R(i,j)$. In the second case, when subsequence $R(i,j)$ is produced by appending $R(j)$ to the optimal folding of $R(i,j-1)$, it can be produced in $S(i,j-1)$ ways.

This is summarized in the recurrence below. For conciseness, we use the Kronecker delta notation $\delta(x,y) = 0$ if $x \neq y$, and $\delta(x,y) = 1$ otherwise.

$$S(i,j) = \sum \begin{cases} \sum\limits_{i \leq k < j \mid R(k) \text{ can be paired with } R(j)} S(i,k-1) \times S(k+1,j-1) \times \\ \qquad \times \delta(V(i,j), V(i,k-1) + V(k+1,j-1) + 1), \\ S(i,j-1) \times \delta(V(i,j), V(i,j-1)). \end{cases} \tag{2}$$

The time complexity of computing all $S(i,j)$ values is $O(n^3 M(n))$: the computation goes over $O(n^2)$ substrings of R, tries $O(n)$ possible ways to construct each and performs at most a multiplication, requiring $O(M(n))$ time. The space complexity for computing $S(i,j)$ is $O(n^3)$, because we have a matrix with $O(n^2)$ cells, and each cell uses $O(n)$ bits.

3.2 The number of optimal global foldings that use an optimal folding of a subsequence

We will now compute a more subtle quantity. Given $i \leq j \in \{1,\ldots,n\}$, we denote by $T(i,j)$ the number of optimal foldings of R that, when restricted to the range from i to j, they are also an optimal folding of the substring $R(i,j)$. In other words, $T(i,j)$ is the total number of global optimal extensions that the optimal foldings of $R(i,j)$ have.

Since we are not considering pseudoknots, bases of R outside of $R(i,j)$ cannot pair with bases in $R(i,j)$. As such, each optimal folding of $R(i,j)$ is independent of the optimal folding of the rest of the string R. As such, if we know $T(i,j)$ and $S(i,j)$, then we can deduce that each optimal folding of $R(i,j)$ can be extended in $T(i,j)/S(i,j)$ ways into a global optimal folding of R. Let us denote the ratio $T(i,j)/S(i,j)$ by $c(i,j)$, and note that, by the above observation, $c(i,j)$ is integer.

The table T will be constructed starting from the full string $R(1,n)$ and moving towards single base substring. By definition, the number of optimal solutions for the full string is the same as the number of optimal solutions that use the full string. Thus, the base case for this algorithm is $T(1,n) = S(1,n)$. All other values of T will be initialized as 0.

The dynamic programming algorithms considered so far have been of collecting type: the required values for computing each dynamic programming cell have been computed earlier in the algorithm. The algorithm for computing T will be of distributing type instead. When this algorithm enters a cell, the value of that cell has already been computed correctly. This value is then used to incrementally derive the correct values of the cells that will be entered later.

When entering a cell (i,j), the algorithm checks all distinct ways how an optimal folding of $R(i,j)$ can be generated from optimal foldings of its substrings. This is done with the same logic as in Nussinov's recurrence (1). Since $T(i,j)$ is already computed correctly, then

Algorithm 1: Computing the table T.

```
1  for each substring R(i,j) in decreasing order of length do
2      c ← T(i,j)/S(i,j);
3      for each way of constructing an optimal folding of R(i,j) do
4          if this optimal folding pairs R(k) with R(j) then
5              a ← c × S(i,k−1) × S(k+1,j−1);
6              T(i,k−1) ← T(i,k−1) + a;
7              T(k+1,j−1) ← T(k+1,j−1) + a;
8          else
               // the optimal folding appends R(j) at the end of the optimal
                  folding for R(i,j−1)
9              a ← c × S(i,j−1);
10             T(i,j−1) ← T(i,j−1) + a;
11             T(j,j) ← T(j,j) + a;
```

we know $c(i,j)$. Namely, we know that each optimal folding of $R(i,j)$ can be extended in $c(i,j)$ ways into a global optimal folding of R. We can then add these ways to each substring of $R(i,j)$ used for generating an optimal folding of $R(i,j)$. Processing the substrings in decreasing order on length guarantees that the final correct value of T for any substring is available by the time that substring is processed.

See Algorithm 1 for the computation of T. For simplicity, in line 3 we write "iterate over each way of constructing an optimal folding of $R(i,j)$" meaning that we iterate over all cases from Nussinov's recurrence (1). That is, we consider all $k \in \{i, \ldots, j-1\}$ such that $R(k)$ can be paired with $R(j)$ and this folding gives an optimal value (i.e., $V(i,j) = V(i,k-1) + V(k+1,j-1) + 1$). This loop also considers the second case of Nussinov's recurrence in which the optimal folding of $R(i,j)$ appends $R(j)$ to the optimal folding for $R(i,j-1)$.

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 may perform a division and a few multiplications requiring $O(M(n))$ time each. As before, the space complexity for computing $T(i,j)$ is $O(n^3)$, because we have a matrix with $O(n^2)$ cells, and each cell uses $O(n)$ bits.

3.3 The number of optimal foldings containing a base 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 substring. In other words, in how many foldings that base appears unpaired. This is useful for computing safety, so let us create a new array U out of the single-base values on the diagonal, having values $U(i) = T(i,i)$.

We will also create a table P, such that $P(i,j)$ is the number of optimal foldings of R that use the pair (i,j). This is not directly available from T, but it can be computed by going over all substrings of R, determining if their optimal foldings use the pair (i,j), and summing the number of optimal global foldings that extend this local folding. This is shown in Algorithm 2. We assume that all $P(i,j)$ cells are initialized as 0.

Algorithm 2: Computing the table R.

```
for each substring R(i, j) do
    c ← T(i, j)/S(i, j);
    for each way of constructing an optimal folding of R(i, j) do
        if this optimal folding pairs R(k) with R(j) then
            P(k, j) ← P(k, j) + c × S(i, k − 1) × S(k + 1, j − 1);
```

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, it performs one or two multiplications requiring $O(M(n))$ time each. As before, the total space complexity is $O(n^3)$.

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 substring $R(i, j)$. 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 U and P in place, it is straightforward to determine the safety of base pairs and unpaired bases in any given folding. A base pair $\{i, j\}$ $(i < j)$ is safe if and only if $P(i, j)$ is equal to the total number of optimal foldings, namely $S(1, n)$. Similarly, an unpaired base i is safe if and only if $U(i) = S(1, n)$.

4 Experimental results

We implemented the safe and complete algorithm of Section 3 in the Go language. To represent big integers we used the standard `math/big` library. Our implementation is available at `https://github.com/kiirala/rna-safe-complete` or through Go environment with command `go get keltainen.duckdns.org/rnafolding`.

We performed experiments to test this implementation on real RNA sequences, mainly focusing on the number of optimal solutions and the running time. Experiments about the biological relevance of the safe solutions can be found in the Appendix. To have a baseline for comparison, we also performed experiments with ViennaRNA's program RNAsubopt [17] implementing the minimum free energy formulation. RNAsubopt computes all optimal and suboptimal foldings that are in a certain range of the optimum. We chose this range to be 1 kcal/mol (option `-e 1`). In order to obtain a safe and complete algorithm for this minimum free energy formulation, given all s optimal and suboptimal solutions reported by RNAsubopt, we applied the trivial safe and complete algorithm mentioned at the beginning of Sec. 3, running in time $O(sn + n^2)$. RNAsubopt's output was piped to this trivial algorithm, so there was no overhead in writing / reading from disk.

We used data from the STRAND RNA database [3]. We analyzed tRNA [7, 32, 25], 5S ribosomal RNA [8, 32], 16S ribosomal RNA [8, 32] and 23S ribosomal RNA [8, 32]. When multiple database entries have exactly the same sequence, only one of such entries is analyzed. Any sequences that are marked as sequence fragments in the database are ignored and only complete sequences are included in the analysis. See Table 1 for the number of sequences analyzed and their average length. Out of the 117 sequences from the 23S rRNA dataset, only on 58 of them RNAsubopt finished running in the allocated time (on some of these 58, it ran for more than 96 hours). As such, all table rows "23S rRNA*" contain results only for these 58 sequences, and the rows "23S rRNA" contain results for all 117 sequences.

Table 1 Characteristics of the four datasets, and statistics on the number of reported solutions by each method. Large values are shown without decimal digits.

		Avg	#Solutions of Max Pairs			#Solutions of RNAsubopt		
	#Inputs	length	Median	Avg	Max	Median	Avg	Max
tRNA	633	78	975	$7 \cdot 10^{11}$	$4 \cdot 10^{14}$	3	6	421
5S rRNA	136	118	37301	$4 \cdot 10^{6}$	$2 \cdot 10^{8}$	10	15	109
16S rRNA	647	1536	$9 \cdot 10^{45}$	$3 \cdot 10^{93}$	$2 \cdot 10^{96}$	$4 \cdot 10^{5}$	$1 \cdot 10^{6}$	$4 \cdot 10^{9}$
23S rRNA*	58	2439	$1 \cdot 10^{86}$	$4 \cdot 10^{99}$	$2 \cdot 10^{101}$	$1 \cdot 10^{7}$	$8 \cdot 10^{7}$	$8 \cdot 10^{8}$
23S rRNA	117	2726	$4 \cdot 10^{89}$	$6 \cdot 10^{130}$	$7 \cdot 10^{132}$	–	–	–

Table 2 Running time (seconds) and average memory usage (MB) by each method. Columns "RNAsubopt – All" refer to RNAsubopt outputting all solutions only. Columns "RNAsubopt – Safe&Compl." include also the trivial safe and complete algorithm. Column "Med" means median.

	Max Pairs – Safe&Compl.				RNAsubopt – All				RNAsubopt – Safe&Compl.			
	Time			Mem.	Time			Mem.	Time			Mem.
	Med	Avg	Max	Max	Med	Avg	Max	Max	Med	Avg	Max	Max
tRNA	0.01	0.01	0.78	41	0.01	0.01	0.37	12	0.01	0.01	0.39	12
5S rRNA	0.04	0.05	0.11	10	0.03	0.04	0.08	10	0.03	0.04	0.1	10
16S rRNA	47	55	310	1964	108	2223	440727	175	131	3199	753752	175
23S rRNA*	256	205	327	2488	4180	25194	281149	273	5454	31457	348989	273
23S rRNA	240	242	931	3561	–	–	–	–	–	–	–	

In Table 1 we also show statistics on the number of optimal solutions to the maximum pairs formulation and the number of optimal and suboptimal solutions reported by RNAsubopt. In both cases, the number of solutions increases exponentially, with a much more rapid growth for the maximum pairs formulation. This also shows that the minimum free energy formulation is more stable, and thus more accurate.

In Table 2 we show the running time and memory usage for: our implementation of the safe and complete algorithm for the maximum pairs formulation, RNAsubopt reporting all optimal and suboptimal solutions, RNAsubopt plus the trivial safe and complete algorithm checking all RNAsubopt's solutions. The datasets tRNA, 5S rRNA and 16S rRNA were run on a machine with an Intel Xeon E5-2697 (2.7 GHz) CPU, with each process limited to 2 GiB memory. The dataset 23S rRNA was run on a machine with an Intel Xeon E3-1220 (3.1 GHz) CPU, with each process limited to 4 GiB memory.

For short RNAs (tRNA and 5S rRNA), the used resources are very similar, with RNAsubopt running slightly faster, likely because it outputs very few solutions. However, for longer RNAs (16S rRNA and 23S rRNA), our safe and complete algorithm for the maximum pairs formulation is significantly faster, even though in our case the number of optimal solutions is significantly larger. Moreover, the running time of RNAsubopt is much more variable for longer RNAs, with some inputs taking more than 96 hours (this particular RNA string had 726 Million solutions reported by RNAsubopt). Our safe and complete algorithm for the maximum pairs formulation finished in all cases in under 16 minutes, even though the numbers of optimal solutions can be of the order 10^{130}. We also observe a larger amount of memory used by our algorithm (up to 3.5GB for 23S rRNA), which is likely due to the fact that the numbers of solutions are much larger. However, this is still not prohibitive on modern machines.

5 Discussion and conclusions

To conclude, we would like to sketch how our technique generalizes to other problems solvable by dynamic programming. Precise algorithms need to be derived for each problem, but the technique introduced in this paper gives a blueprint for obtaining such algorithms. To make this more formal, suppose we have a dynamic programming recurrence of the form:

$$V(i,j) = \bigoplus_{\ell \in L_{i,j}} \left\{ f_\ell\big(V(i_1^\ell, j_1^\ell), \ldots, V(i_{k_\ell}^\ell, j_{k_\ell}^\ell)\big) \mid \varphi_\ell(i_1^\ell, j_1^\ell, \ldots, i_{k_\ell}^\ell, j_{k_\ell}^\ell) \right\} \tag{3}$$

where $L_{i,j}$ is some set of possible cases to consider in computing $V(i,j)$, $\bigoplus$ is an operation of these $|L_{i,j}|$ cases, for example min or max, values $V(i_1^\ell, j_1^\ell), \ldots, V(i_{k_\ell}^\ell, j_{k_\ell}^\ell)$ have previously been computed when computing $V(i,j)$, f_ℓ is a function on these previously computed values (for example sum), and φ_ℓ is a condition that must hold in order to consider the ℓ-th case in the recurrence. We also require that each of these $|L_{i,j}|$ cases leads to a different optimal solution (we observed at the beginning of Sec. 3.1 that this also holds for Nussinov's recurrence). It is immediate to verify that Nussinov's recurrence (1) fits into the above recurrence scheme (3). Note also that, even though recurrence (3) has only two parameters (i,j), our technique works for an arbitrary number of parameters.

Having all values $V(i,j)$ computed, one can obtain a safe and complete algorithm that can detect which cases allowed by the sets $L_{i,j}$ and the functions φ_ℓ are present in all optimal solutions. We need to apply exactly the same three steps outlined in Sec. 3. First, we need to compute the matrix $S(i,j)$ counting the number of optimal solutions of the partial input (i,j), then we need to compute the matrix $T(i,j)$ counting the number of optimal full solutions that, when restricted to (i,j), they are also an optimal solution for the partial input (i,j). Finally, one can compute a matrix analogous to our $P(i,j)$, which counts, for every possible partial solution that is a candidate to be safe, in how many cases of the recurrence it appears, and how many full optimal solutions these give rise to (information available from matrix T). These counts then indicate in how many full optimal solutions each partial solution appears, and thus indicate safety.

In this paper we applied our technique to the classical RNA folding recurrence of Nussinov. The choice of this problem was motivated by two factors: (1) it is simple enough to illustrate our technique and its general applicability, and (2) it is also a problem which on real data admits a large number of solutions, making the notion of safety practically relevant. For example, we observed RNA sequences admitting an exponential number of solutions. Despite this, our implementation of the safe and complete algorithm for it was efficient and ran in 16 minutes at most. Moreover, our experiments from the Appendix also show that safe sub-solutions for the maximum pairs formulation match the true biological folding with a precision of more than 40%.

However, we should also note that in the case of RNA folding, the minimum free energy model and its refinements are more biologically accurate. Our experiments on combining the output of RNAsubopt and the trivial algorithm for computing safety (in the Appendix) indicate that the safe sub-solutions for the minimum energy formulation have a significantly higher precision than for the maximum pairs formulation, reflecting the more biological accurate problem formulation. Thus, it would be interesting to derive efficient safe and complete algorithms for the minimum free energy model. Such algorithms would also need to be compared to McCaskill [19] approach deriving "predominance" of RNA folding substructures based on partition-function based methods.

References

1 Tatsuya Akutsu. Approximation and Exact Algorithms for RNA Secondary Structure Prediction and Recognition of Stochastic Context-free Languages. *Journal of Combinatorial Optimization*, 3(2):321–336, July 1999. doi:10.1023/A:1009898029639.

2 Tatsuya Akutsu. Dynamic programming algorithms for RNA secondary structure prediction with pseudoknots. *Discrete Applied Mathematics*, 104(1):45–62, 2000. doi:10.1016/S0166-218X(00)00186-4.

3 Mirela Andronescu, Vera Bereg, Holger H Hoos, and Anne Condon. RNA STRAND: the RNA secondary structure and statistical analysis database. *BMC Bioinformatics*, 9(1):340, 2008. doi:10.1186/1471-2105-9-340.

4 Masanori Arita. Graph modeling of metabolism. *Journal-Japanese Society for Artificial Intelligence*, 15(4):703–710, 2000.

5 Masanori Arita. Metabolic reconstruction using shortest paths. *Simulation Practice and Theory*, 8(1-2):109–125, 2000.

6 J. K. Baker. Trainable grammars for speech recognition. *The Journal of the Acoustical Society of America*, 65(S1):S132–S132, 1979. doi:10.1121/1.2017061.

7 Helen M. Berman, Wilma K. Olson, David L. Beveridge, John Westbrook, Anke Gelbin, Tamas Demeny, Shu-Hsin Hsieh, A.R. Srinivasan, and Bohdan Schneider. The nucleic acid database. A comprehensive relational database of three-dimensional structures of nucleic acids. *Biophys J*, 63(3):751–759, 1992.

8 Jamie J. Cannone, Sankar Subramanian, Murray N. Schnare, James R. Collett, Lisa M. D'Souza, Yushi Du, Brian Feng, Nan Lin, Lakshmi V. Madabusi, Kirsten M. Müller, Nupur Pande, Zhidi Shang, Nan Yu, and Robin R. Gutell. The Comparative RNA Web (CRW) Site: an online database of comparative sequence and structure information for ribosomal, intron, and other RNAs. *BMC Bioinformatics*, 3(1):2, January 2002. doi:10.1186/1471-2105-3-2.

9 Kun-Mao Chao, Ross C. Hardison, and Webb Miller. Locating well-conserved regions within a pairwise alignment. *CABIOS*, 9(4):387–396, 1993. doi:10.1093/bioinformatics/9.4.387.

10 Richard Durbin, Sean R Eddy, Anders Krogh, and Graeme Mitchison. *Biological sequence analysis: probabilistic models of proteins and nucleic acids*. Cambridge university press, 1998.

11 David Eppstein. K-Best Enumeration. *Bulletin of the EATCS*, 115, 2015. URL: http://eatcs.org/beatcs/index.php/beatcs/article/view/322.

12 David Eppstein. k-Best Enumeration. In *Encyclopedia of Algorithms*. Springer, Berlin, Heidelberg, 2015. doi:10.1007/978-3-642-27848-8_733-1.

13 Yelena Frid and Dan Gusfield. A simple, practical and complete O-time Algorithm for RNA folding using the Four-Russians Speedup. *Algorithms for Molecular Biology*, 5(1):13, January 2010. doi:10.1186/1748-7188-5-13.

14 Andreas Friemann and Stefan Schmitz. A new approach for displaying identities and differences among aligned amino acid sequences. *Comput Appl Biosci*, 8(3):261–265, June 1992.

15 Robert Giegerich, Carsten Meyer, and Peter Steffen. A discipline of dynamic programming over sequence data. *Science of Computer Programming*, 51(3):215–263, 2004. doi:10.1016/j.scico.2003.12.005.

16 Ivo L. Hofacker, Peter Schuster, and Peter F. Stadler. Combinatorics of RNA secondary structures. *Discrete Applied Mathematics*, 88(1):207–237, 1998. Computational Molecular Biology DAM - CMB Series. doi:10.1016/S0166-218X(98)00073-0.

17 Ronny Lorenz, Stephan H Bernhart, Christian Höner zu Siederdissen, Hakim Tafer, Christoph Flamm, Peter F Stadler, and Ivo L Hofacker. ViennaRNA package 2.0. *Algorithms for Molecular Biology*, 6(26), November 2011. doi:10.1186/1748-7188-6-26.

18 David H Mathews, Jeffrey Sabina, Michael Zuker, and Douglas H Turner. Expanded sequence dependence of thermodynamic parameters improves prediction of RNA secondary structure. *Journal of Molecular Biology*, 288(5):911–940, 1999. doi:10.1006/jmbi.1999.2700.

19 John S. McCaskill. The Equilibrium Partition Function and Base Pair Binding Probabilities for RNA Secondary Structure. *Biopolymers*, 29(6-7):1105–1119, 1990. doi:10.1002/bip.360290621.

20 Niranjan Nagarajan and Mihai Pop. Parametric Complexity of Sequence Assembly: Theory and Applications to Next Generation Sequencing. *Journal of Computational Biology*, 16(7):897–908, 2009.

21 Ruth Nussinov and Ann B. Jacobson. 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):6309–6313, November 1980. doi:10.1073/pnas.77.11.6309.

22 Leena Salmela and Alexandru I. Tomescu. Safely Filling Gaps with Partial Solutions Common to All Solutions. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 2019. doi:10.1109/TCBB.2017.2785831.

23 Arnold Schönhage and Volker Strassen. Schnelle Multiplikation großer Zahlen. *Computing*, 7(3-4):281–292, 1971. doi:10.1007/BF02242355.

24 Yu-Keng Shih and Srinivasan Parthasarathy. A single source k-shortest paths algorithm to infer regulatory pathways in a gene network. *Bioinformatics*, 28(12):i49–i58, 2012.

25 Mathias Sprinzl and Konstantin S. Vassilenko. Compilation of tRNA sequences and sequences of tRNA genes. *Nucleic Acids Research*, 33(Database issue):139–140, 2005.

26 Alexandru I. Tomescu and Paul Medvedev. Safe and Complete Contig Assembly Through Omnitigs. *Journal of Computational Biology*, 24(6):590–602, 2017. doi:10.1089/cmb.2016.0141.

27 Leslie G. Valiant. General context-free recognition in less than cubic time. *Journal of Computer and System Sciences*, 10(2):308–315, 1975. doi:10.1016/S0022-0000(75)80046-8.

28 Martin Vingron. Near-optimal sequence alignment. *Curr. Opinion in Structural Biol.*, 6(3):346–352, June 1996. doi:10.1016/s0959-440x(96)80054-6.

29 Martin Vingron and Patrick Argos. Determination of reliable regions in protein sequence alignments. *Prot. Engin.*, 3(7):565–569, 1990. doi:10.1093/protein/3.7.565.

30 Amy E Walter, Douglas H Turner, James Kim, Matthew H Lyttle, Peter Müller, David H Mathews, and Michael Zuker. Coaxial stacking of helixes enhances binding of oligoribonucleotides and improves predictions of RNA folding. *Proceedings of the National Academy of Sciences*, 91(20):9218–9222, 1994. doi:10.1073/pnas.91.20.9218.

31 Michael S. Waterman and Thomas H. Byers. A dynamic programming algorithm to find all solutions in a neighborhood of the optimum. *Mathematical Biosciences*, 77(1):179–188, 1985. doi:10.1016/0025-5564(85)90096-3.

32 John Westbrook, Zukang Feng, Li Chen, Huanwang Yang, and Helen M. Berman. The Protein Data Bank and structural genomics. *Nucleic Acids Research*, 31(1):489–491, January 2003. doi:10.1093/nar/gkg068.

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

34 Shay Zakov, Dekel Tsur, and Michal Ziv-Ukelson. Reducing the worst case running times of a family of RNA and CFG problems, using Valiant's approach. *Algorithms for Molecular Biology*, 6(1):20, August 2011. doi:10.1186/1748-7188-6-20.

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

36 Michael Zuker. Suboptimal sequence alignment in molecular biology: Alignment with error analysis. *J Mol Biol*, 221(2):403–420, September 1991.

A Biological relevance of the notion of safety

We also compared how well the safe sub-solutions match the reference biological foldings from the STRAND RNA database. Each possible $\binom{n}{2}$ base pairs $\{i, j\}$, with $1 \leq i < j \leq n$, and each unpaired base i, with $1 \leq i \leq n$, was classified as:

- true positive (TP), if it is declared safe, and it appears in the reference folding,
- false positive (FP), if it is declared safe, but it does not appear in the reference folding,
- false negative (FN), if it is not declared safe, but it appears in the reference folding.

For each folding and each method (maximum pairs and minimum free energy), we computed the numbers TP, FP, FN. Then, we computed precision, as TP/(TP+FP), and recall, as TP/(TP+FN). Intuitively, precision measures how correct the safe sub-solutions are, and recall measures how much of the reference folding is correctly classified as safe. We also computed the proportion of all bases in the input RNA strings that are classified as safe. We report summary statistics on these metrics in Table 3. In Figure 2 we plot the precision and recall for each individual molecule.

In order to analyze how much the notion of safe sub-solution improves the correctness of the solutions, we also computed the precision and recall metrics when considering an entire solution as "safe" in the above definitions of TP, FP and FN. For this experiment, we ran our maximum pairs implementation and RNAsubopt so that they report a single optimal solution. We show these results in Table 4 and in Figure 2. Note that Figure 2 shows that all points for the single solutions lie close to the diagonal. This is due to the predicted solutions having roughly the same amount of paired bases as the biologically correct solutions and thus TP+FP is roughly equal to TP+FN.

Comparing Tables 3 and 4, we observe that for both methods, the notion of safe sub-solution is relevant, since it generally improves precision at the cost of recall. Figure 2 shows that the safety notion moves the points towards right. The increase in precision is more pronounced for the maximum pairs formulations, e.g., with an increase in median precision between 37% and 72% in the four data sets as shown in Table 5.

In our experiments, precision is not affected by the proportion of safe decisions. However, Figure 3 shows that high recall correlates with a high proportion of safe decisions. These observations hold for both our safe and complete algorithm and for RNAsubopt. For the sequences with a high proportion of safe decisions, the optimal folding tends to be unambiguous, which results in increased recall.

The maximum pairs model has a smaller proportion of safe sub-solutions than the minimum free energy model, because it admits significantly more solutions. As such, also its recall is lower. Table 3 and Figure 2 show that the precision and recall metrics of the maximum pairs model are lower than for the minimum free energy model. This is a consequence of the fact that the minimum free energy model is more biologically accurate.

Table 3 The precision and recall of the safe sub-solutions reported by each method, for all molecules in the datasets. We also show the proportion of bases of the RNA strings that are classified as safe.

	Max Pairs – Safe and Complete						RNAsubopt – Safe and Complete					
	Precision		Recall		Prop. Safe		Precision		Recall		Prop. Safe	
	Med	Avg	Med	Avg	Med	Avg	Med	Avg	Med	Avg	Med	Avg
tRNA	0.68	0.67	0.19	0.21	0.33	0.39	0.89	0.86	0.75	0.72	0.87	0.83
5S rRNA	0.58	0.53	0.11	0.15	0.24	0.28	0.84	0.79	0.65	0.59	0.83	0.76
16S rRNA	0.37	0.42	0.07	0.07	0.20	0.23	0.60	0.60	0.33	0.33	0.59	0.58
23S rRNA*	0.40	0.42	0.06	0.07	0.17	0.20	0.66	0.64	0.38	0.38	0.62	0.61
23S rRNA	0.41	0.43	0.06	0.06	0.15	0.18	–	–	–	–	–	–

Table 4 The precision and recall of a single solution reported by each method, for all molecules in the datasets.

	Max Pairs – Single				RNAsubopt – Single			
	Precision		Recall		Precision		Recall	
	Med	Avg	Med	Avg	Med	Avg	Med	Avg
tRNA	0.45	0.45	0.41	0.41	0.83	0.80	0.85	0.82
5S rRNA	0.34	0.36	0.30	0.33	0.76	0.71	0.75	0.70
16S rRNA	0.27	0.27	0.23	0.24	0.49	0.49	0.47	0.47
23S rRNA	0.28	0.29	0.25	0.25	0.54	0.55	0.53	0.52

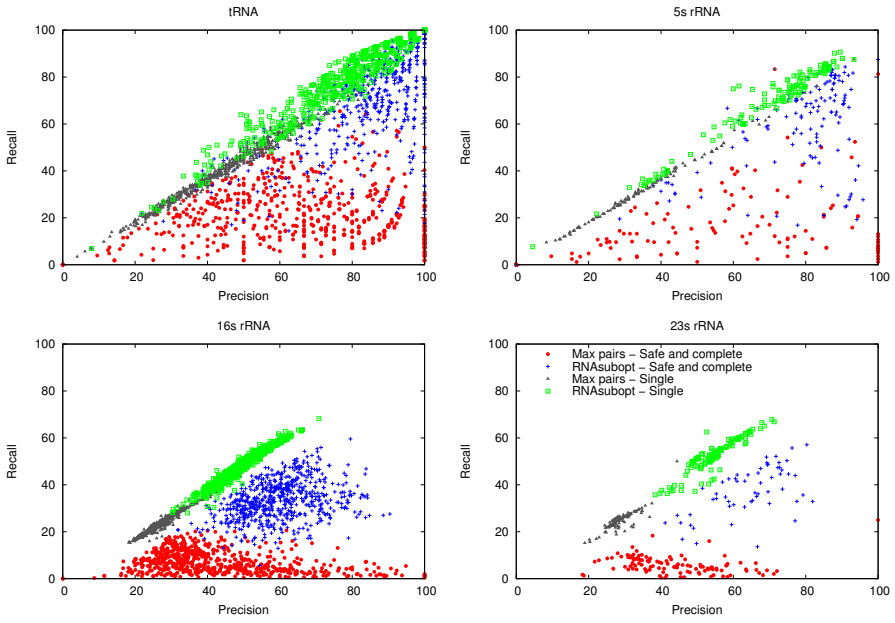


Figure 2 The precision and recall of each molecule in the datasets.

Table 5 The relative increase of precision and recall of the safe sub-solutions, as compared to a single solution, for all molecules in the datasets.

	Max Pairs				RNAsubopt			
	Precision		Recall		Precision		Recall	
	Med	Avg	Med	Avg	Med	Avg	Med	Avg
tRNA	52%	50%	-52%	-48%	8%	6%	-12%	-12%
5S rRNA	72%	49%	-65%	-54%	10%	11%	-13%	-16%
16S rRNA	37%	54%	-72%	-70%	22%	22%	-30%	-30%
23S rRNA	41%	47%	-75%	-73%	22%	17%	-28%	-28%

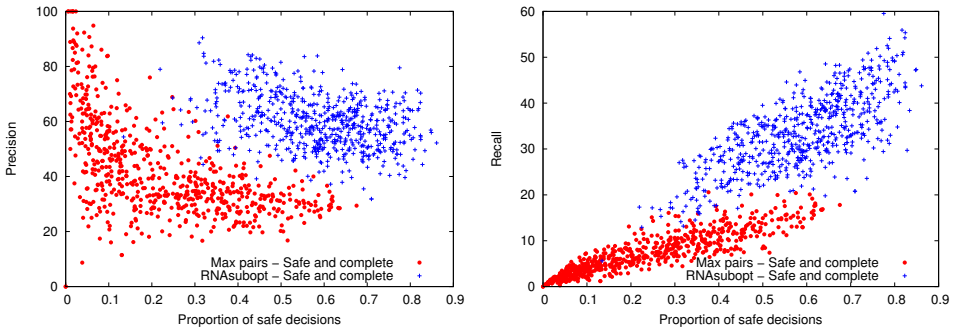


Figure 3 Precision and recall as a function of the proportion of safe decisions for each molecule in the 16S rRNA dataset.