ALGORITHMS FOR MOLECULAR BIOLOGY

RESEARCH Open Access

# Reducing the worst case running times of a family of RNA and CFG problems, using Valiant's approach

Shay Zakov, Dekel Tsur and Michal Ziv-Ukelson*

**Abstract**

**Background:** RNA secondary structure prediction is a mainstream bioinformatic domain, and is key to computational analysis of functional RNA. In more than 30 years, much research has been devoted to defining different variants of RNA structure prediction problems, and to developing techniques for improving prediction quality. Nevertheless, most of the algorithms in this field follow a similar dynamic programming approach as that presented by Nussinov and Jacobson in the late 70's, which typically yields cubic worst case running time algorithms. Recently, some algorithmic approaches were applied to improve the complexity of these algorithms, motivated by new discoveries in the RNA domain and by the need to efficiently analyze the increasing amount of accumulated genome-wide data.

**Results:** We study Valiant's classical algorithm for Context Free Grammar recognition in sub-cubic time, and extract features that are common to problems on which Valiant's approach can be applied. Based on this, we describe several problem templates, and formulate generic algorithms that use Valiant's technique and can be applied to all problems which abide by these templates, including many problems within the world of RNA Secondary Structures and Context Free Grammars.

**Conclusions:** The algorithms presented in this paper improve the theoretical asymptotic worst case running time bounds for a large family of important problems. It is also possible that the suggested techniques could be applied to yield a practical speedup for these problems. For some of the problems (such as computing the RNA partition function and base-pair binding probabilities), the presented techniques are the only ones which are currently known for reducing the asymptotic running time bounds of the standard algorithms.

## 1 Background

RNA research is one of the classical domains in bioinformatics, receiving increasing attention in recent years due to discoveries regarding RNA's role in regulation of genes and as a catalyst in many cellular processes [1,2]. It is well-known that the function of an RNA molecule is heavily dependent on its structure [3]. However, due to the difficulty of *physically* determining RNA structure via wet-lab techniques, *computational prediction* of RNA structures serves as the basis of many approaches related to RNA functional analysis [4]. Most computational tools for RNA structural prediction focus on RNA *secondary structures* - a reduced structural representation of RNA molecules which describes a set of paired nucleotides, through hydrogen bonds, in an RNA sequence. RNA secondary structures can be relatively well predicted computationally in polynomial time (as opposed to three-dimensional structures). This computational feasibility, combined with the fact that RNA secondary structures still reveal important information about the functional behavior of RNA molecules, account for the high popularity of state-of-the-art tools for RNA secondary structure prediction [5].

Over the last decades, several variants of RNA secondary structure prediction problems were defined, to which polynomial algorithms have been designed. These variants include the basic *RNA folding* problem (predicting the secondary structure of a single RNA strand which is given as an input) [6-8], the *RNA-RNA*

* Correspondence: michaluz@cs.bgu.ac.il
Department of Computer Science, Ben-Gurion University of the Negev, P.O.B. 653 Beer Sheva, 84105, Israel

*Interaction* problem (predicting the structure of the complex formed by two or more interacting RNA molecules) [9], the *RNA Partition Function and Base Pair Binding Probabilities* problem of a single RNA strand [10] or an RNA duplex [11,12] (computing the pairing probability between each pair of nucleotides in the input), the *RNA Sequence to Structured-Sequence Alignment* problem (aligning an RNA sequence to sequences with known structures) [13,14], and the *RNA Simultaneous Alignment and Folding* problem (finding a secondary structure which is conserved by multiple homologous RNA sequences) [15]. Sakakibara et al. [16] noticed that the basic *RNA Folding* problem is in fact a special case of the *Weighted Context Free Grammar (WCFG) Parsing* problem (also known as *Stochastic* or *Probabilistic CFG Parsing*) [17]. Their approach was then followed by Dowell and Eddy [18], Do et al. [19], and others, who studied different aspects of the relationship between these two domains. The WCFG Parsing problem is a generalization of the simpler non-weighted *CFG Parsing* problem. Both WCFG and CFG Parsing problems can be solved by the Cocke-Kasami-Younger (CKY) dynamic programming algorithm [20-22], whose running time is cubic in the number of words in the input sentence (or in the number of nucleotides in the input RNA sequence).

The CFG literature describes two improvements which allow to obtain a sub-cubic time for the CKY algorithm. The first among these improvements was a technique suggested by Valiant [23], who showed that the CFG Parsing problem on a sentence with $n$ words can be solved in a running time which matches the running time of a *Boolean Matrix Multiplication* of two $n \times n$ matrices. The current asymptotic running time bound for this variant of matrix multiplication was given by Coppersmith-Winograd [24], who showed an $O(n^{2.376})$ time (theoretical) algorithm. In [25], Akutsu argued that the algorithm of Valiant can be modified to deal also with WCFG Parsing (this extension is described in more details in [26]), and consequentially with RNA Folding. The running time of the adapted algorithm is different from that of Valiant's algorithm, and matches the running time of a *Max-Plus Multiplication* of two $n \times n$ matrices. The current running time bound for this variant is $O\left(\frac{n^3 \log^3 \log n}{\log^2 n}\right)$, given by Chan [27].

The second improvement to the CKY algorithm was introduced by Graham et al. [28], who applied the *Four Russians* technique [29] and obtained an $O\left(\frac{n^3}{\log n}\right)$ running time algorithm for the (non-weighted) CFG Parsing problem. To the best of our knowledge, no extension of this approach to the WCFG Parsing problem has been described. Recently, Frid and Gusfield [30] showed how to apply the *Four Russians* technique to the RNA folding problem (under the assumption of a discrete scoring scheme), obtaining the same running time of $O\left(\frac{n^3}{\log n}\right)$. This method was also extended to deal with the RNA simultaneous alignment and folding problem [31], yielding an $O\left(\frac{n^6}{\log n}\right)$ running time algorithm.

Several other techniques have been previously developed to accelerate the practical running times of different variants of CFG and RNA related algorithms. Nevertheless, these techniques either retain the same worst case running times of the standard algorithms [14,28,32-36], or apply heuristics which compromise the optimality of the obtained solutions [25,37,38]. For some of the problem variants, such as the *RNA Base Pair Binding Probabilities* problem (which is considered to be one of the variants that produces more reliable predictions in practice), no speedup to the standard algorithms has been previously described.

In his paper [23], Valiant suggested that his approach could be extended to additional related problems. However, in more than three decades which have passed since then, very few works have followed. The only extension of the technique which is known to the authors is Akutsu's extension to WCFG Parsing and RNA Folding [25,26]. We speculate that simplifying Valiant's algorithm would make it clearer and thus more accessible to be applied to a wider range of problems.

Indeed, in this work we present a simple description of Valiant's technique, and then further generalize it to cope with additional problem variants which do not follow the standard structure of CFG/WCFG Parsing (a preliminary version of this work was presented in [39]). More specifically, we define three template formulations, entitled *Vector Multiplication Templates (VMTs)*. These templates abstract the essential properties that characterize problems for which a Valiant-like algorithmic approach can yield algorithms of improved time complexity. Then, we describe generic algorithms for all problems sustaining these templates, which are based on Valiant's algorithm.

Table 1 lists some examples of VMT problems. The table compares between the running times of standard dynamic programming (DP) algorithms, and the VMT algorithms presented here. In the single string problems, $n$ denotes the length of the input string. In the double-string problems [9,12,13], both input strings are assumed to be of the same length $n$. For the *RNA Simultaneous Alignment and Folding* problem, $m$ denotes the number of input strings and $n$ is the length of each string. $DB(n)$ denotes the time complexity of a Dot Product or a Boolean Multiplication of two $n \times n$ matrices, for which the current best theoretical result is

**Table 1 Time complexities of several VMT problems**

| | Problem | Standard DP running time | Implicit [explicit] VMT algorithm running time |
|---|---|---|---|
| Results previously published | CFG Recognition/Parsing | $\Theta(n^3)$ [20-22] | $\Theta(DB(n))\ [\Theta(n^{2.38})]$[23] |
| | WCFG Parsing | $\Theta(n^3)$[17] | $\Theta(MP(n))\left[\tilde{O}\left(\frac{n^3}{\log^2 n}\right)\right]$[25] |
| | RNA Single Strand Folding | $\Theta(n^3)$[6,7] | $\Theta(MP(n))\left[\tilde{O}\left(\frac{n^3}{\log^2 n}\right)\right]$[25] |
| | RNA Partition Function | $\Theta(n^3)$[10] | $\Theta(MP(n))\ [\Theta(n^{2.38})]$[25] |
| In this paper | WCFG Inside-Outside | $\Theta(n^3)$[43] | $\Theta(DB(n))\ [\Theta(n^{2.38})]$ |
| | RNA Base Pair Binding Probabilities | $\Theta(n^3)$[10] | $\Theta(DB(n))\ [\Theta(n^{2.38})]$ |
| | RNA Simultaneous Alignment and Folding | $\Theta((n/2)^{3m})$[15] | $\Theta(MP(n^m))\left[\tilde{O}\left(\frac{n^3 m}{m \log^2 n}\right)\right]$ |
| | RNA-RNA Interaction | $\Theta(n^6)$[9] | $\Theta(MP(n^2))\left[\tilde{O}\left(\frac{n^6}{\log^2 n}\right)\right]$ |
| | RNA-RNA Interaction Partition Function | $\Theta(n^6)$[12] | $\Theta(DB(n))\ [\Theta(n^{4.75})]$ |
| | RNA Sequence to Structured-Sequence Alignment | $\Theta(n^4)$[13] | $\Theta(nMP(n))\left[\tilde{O}\left(\frac{n^4}{\log^2 n}\right)\right]$ |

The notation $\tilde{O}$ corresponds to the standard big-*O* notation, hiding some polylogarithmic factors.

$O(n^{2.376})$, due to Coppersmith and Winograd [24]. *MP* (*n*) denotes the time complexity of a Min-Plus or a Max-Plus Multiplication of two $n \times n$ matrices, for which the current best theoretical result is $O\left(\frac{n^3 \log^3 \log n}{\log^2 n}\right)$, due to Chan [27]. For most of the problems, the algorithms presented here obtain lower running time bounds than the best algorithms previously known for these problems. It should be pointed out that the above mentioned matrix multiplication running times are the theoretical asymptotic times for sufficiently large matrices, yet they do not reflect the actual multiplication time for matrices of realistic sizes. Nevertheless, practical fast matrix multiplication can be obtained by using specialized hardware [40,41] (see Section 6).

The formulation presented here has several advantages over the original formulation in [23]: First, it is considerably simpler, where the correctness of the algorithms follows immediately from their descriptions. Second, some requirements with respect to the nature of the problems that were stated in previous works, such as operation commutativity and distributivity requirements in [23], or the *semiring* domain requirement in [42], can be easily relaxed. Third, this formulation applies in a natural manner to algorithms for several classes of problems, some of which we show here. Additional problem variants which do not follow the exact templates presented here, such as the formulation in [12] for the RNA-RNA Interaction Partition Function problem, or the formulation in [13] for the RNA Sequence to Structured-Sequence Alignment problem, can be solved by introducing simple modifications to the algorithms we present. Interestingly, it turns out that almost every variant of RNA secondary structure prediction problem, as well as additional problems from the domain of CFGs, sustain the VMT requirements. Therefore, Valiant's technique can be applied to reduce the worst case running times of a large family of important problems. In general, as explained later in this paper, VMT problems are characterized in that their computation requires the execution of many vector multiplication operations, with respect to different multiplication variants (*Dot Product*, *Boolean Multiplication*, and *Min/Max Plus Multiplication*). Naively, the time complexity of each vector multiplication is linear in the length of the multiplied vectors. Nevertheless, it is possible to organize these vector multiplications as parts of square matrix multiplications, and to apply fast matrix multiplication algorithms in order to obtain a sub-linear (amortized) running time for each vector multiplication. As we show, a main challenge in algorithms for VMT problems is to describe how to bundle subsets of vector multiplications operations in order to compute them via the application of fast matrix multiplication algorithms. As the elements of these vectors are computed along the run of the algorithm, another aspect which requires attention is the decision of the order in which these matrix multiplications take place.

**Road Map**

In Section 2 the basic notations are given. In Section 3 we describe the *Inside Vector Multiplication Template* - a template which extracts features for problems to which Valiant's algorithm can be applied. This section also includes the description of an exemplary problem (Section 3.1), and a generalized and simplified exhibition of Valiant's algorithm and its running time analysis (Section 3.3). In Sections 4 and 5 we define two additional problem templates: the *Outside Vector Multiplication Template*, and the *Multiple String Vector Multiplication Template*, and describe modifications to the algorithm of Valiant which allow to solve problems that sustain these templates. Section 6 concludes the paper, summarizing the main results and discussing some of its implications. Two additional exemplary problems (an Outside and a Multiple String VMT problems) are presented in the Appendix.

## 2 Preliminaries

As intervals of integers, matrices, and strings will be extensively used throughout this work, we first define some related notation.

### 2.1 Interval notations

For two integers $a$, $b$, denote by $[a, b]$ the interval which contains all integers $q$ such that $a \leq q \leq b$. For two intervals $I = [i_1, i_2]$ and $J = [j_1, j_2]$, define the following intervals: $[I, J] = \{q : i_1 \leq q \leq j_2\}$, $(I, J) = \{q : i_2 < q < j_1\}$, $[I, J) = \{q : i_1 \leq q < j_1\}$, and $(I, J] = \{q : i_2 < q \leq j_2\}$ (Figure 1). When an integer $r$ replaces one of the intervals $I$ or $J$ in the notation above, it is regarded as the interval $[r, r]$. For example, $[0, I) = \{q : 0 \leq q < i_1\}$, and $(i, j) = \{q : i < q < j\}$. For two intervals $I = [i_1, i_2]$ and $J = [j_1, j_2]$ such that $j_1 = i_2 + 1$, define $IJ$ to be the concatenation of $I$ and $J$, i.e. the interval $[i_1, j_2]$.

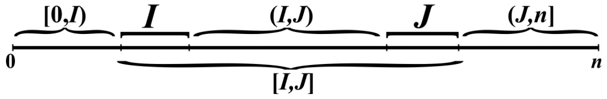


**Figure 1 Some interval examples**.

### 2.2 Matrix notations

Let $X$ be an $n_1 \times n_2$ matrix, with rows indexed with 0, 1, ..., $n_1$ - 1 and columns indexed with 0, 1, ..., $n_2$ - 1. Denote by $X_{i,j}$ the element in the $i$th row and $j$th column of $X$. For two intervals $I \subseteq [0, n_1)$ and $J \subseteq [0, n_2)$, let $X_{I,J}$ denote the sub-matrix of $X$ obtained by projecting it onto the subset of rows $I$ and subset of columns $J$. Denote by $X_{i,J}$ the sub-matrix $X_{[i,i],J}$, and by $X_{I,j}$ the sub-matrix $X_{I,[j,j]}$. Let $\mathcal{D}$ be a domain of elements, and $\otimes$ and $\oplus$ be two binary operations on $\mathcal{D}$. We assume that (1) $\oplus$ is associative (i.e. for three elements $a$, $b$, $c$ in the domain, $(a \oplus b) \oplus c = a \oplus (b \oplus c)$), and (2) there exists a *zero* element $\varphi$ in $\mathcal{D}$, such that for every element $a \in \mathcal{D} a \oplus \varphi = \varphi \oplus a = a$ and $a \otimes \varphi = \varphi \otimes a = \varphi$.

Let $X$ and $Y$ be a pair of matrices of sizes $n_1 \times n_2$ and $n_2 \times n_3$, respectively, whose elements are taken from $\mathcal{D}$. Define the result of the *matrix multiplication* $X \otimes Y$ to be the matrix $Z$ of size $n_1 \times n_3$, where each entry $Z_{i,j}$ is given by

$$Z_{i,j} = \oplus_{q \in [0,n_2)}(X_{i,q} \otimes Y_{q,j}) = (X_{i,0} \otimes Y_{0,j}) \oplus (X_{i,1} \otimes Y_{1,j}) \oplus \ldots \oplus (X_{i,n_2-1} \otimes Y_{n_2-1,j}).$$

In the special case where $n_2 = 0$, define the result of the multiplication $Z$ to be an $n_1 \times n_3$ matrix in which all elements are $\varphi$. In the special case where $n_1 = n_3 = 1$, the matrix multiplication $X \otimes Y$ is also called a *vector multiplication* (where the resulting matrix $Z$ contains a single element).

Let $X$ and $Y$ be two matrices. When $X$ and $Y$ are of the same size, define the result of the *matrix addition* $X \oplus Y$ to be the matrix $Z$ of the same size as $X$ and $Y$, where $Z_{i,j} = X_{i,j} \oplus Y_{i,j}$. When $X$ and $Y$ have the same number of columns, denote by $\begin{bmatrix} X \\ Y \end{bmatrix}$ the matrix obtained by concatenating $Y$ below $X$. When $X$ and $Y$ have the same number of rows, denote by $[XY]$ the matrix obtained by concatenating $Y$ to the right of $X$. The following properties can be easily deduced from the definition of matrix multiplication and the associativity of the $\oplus$ operation (in each property the participating matrices are assumed to be of the appropriate sizes).

$$\begin{bmatrix} X^1 \\ X^2 \end{bmatrix} \otimes Y = \begin{bmatrix} X^1 \otimes Y \\ X^2 \otimes Y \end{bmatrix} \quad (1)$$

$$X \otimes [Y^1 Y^2] = [(X \otimes Y^1)(X \otimes Y^2)] \quad (2)$$

$$(X^1 \otimes Y^1) \oplus (X^2 \otimes Y^2) = [X^1 X^2] \otimes \begin{bmatrix} Y^1 \\ Y^2 \end{bmatrix} \quad (3)$$

Under the assumption that the operations $\otimes$ and $\oplus$ between two domain elements consume $\Theta(1)$ computation time, a straightforward implementation of a matrix multiplication between two $n \times n$ matrices can be computed in $\Theta(n^3)$ time. Nevertheless, for some variants of multiplications, sub-cubic algorithms for square matrix multiplications are known. Here, we consider three such variants, which will be referred to as *standard multiplications* in the rest of this paper:

- *Dot Product*: The matrices hold numerical elements, $\otimes$ stands for number multiplication ($\cdot$) and $\oplus$ stands for number addition (+). The *zero* element is 0. The running time of the currently fastest algorithm for this variant is $O(n^{2.376})$ [24].
- *Min-Plus/Max-Plus Multiplication*: The matrices hold numerical elements,$\otimes$ stands for number addition and $\oplus$ stands for min or max (where $a$ min $b$ is the minimum between $a$ and $b$, and similarly for max). The *zero* element is $\infty$ for the Min-Plus variant and $-\infty$ for the Max-Plus variant. The running time of the currently fastest algorithm for these variants is $O\left(\frac{n^3 \log^3 \log n}{\log^2 n}\right)$[27].
- *Boolean Multiplication*: The matrices hold boolean elements, $\otimes$ stands for *boolean AND* ($\wedge$) and $\oplus$ stands for *boolean OR*($\vee$). The *zero* element is the *false* value. Boolean Multiplication is computable with the same complexity as the Dot Product, having the running time of $O(n^{2.376})$ [24].

## 2.3 String notations

Let $s = s_0 s_1 \ldots s_{n-1}$ be a string of length $n$ over some alphabet. A *position* $q$ in $s$ refers to a point between the characters $s_{q-1}$ and $s_q$ (a position may be visualized as a vertical line which separates between these two characters). Position 0 is regarded as the point just before $s_0$, and position $n$ as the point just after $s_{n-1}$. Denote by $||s|| = n + 1$ the number of different positions in $s$. Denote by $s_{i,j}$ the substring of $s$ between positions $i$ and $j$, i.e. the string $s_i s_{i+1} \ldots s_{j-1}$. In a case where $i = j$, $s_{i,j}$ corresponds to an empty string, and for $i > j$, $s_{i,j}$ does not correspond to a valid string.

An *inside property* $\beta_{i,j}$ is a property which depends only on the substring $s_{i,j}$ (Figure 2). In the context of RNA, an input string usually represents a sequence of nucleotides, where in the context of CFGs, it usually represents a sequence of words. Examples of inside properties in the world of RNA problems are the maximum number of base-pairs in a secondary structure of $s_{i,j}$ [6], the minimum free energy of a secondary structure of $s_{i,j}$ [7], the sum of weights of all secondary structures of $s_{i,j}$ [10], etc. In CFGs, inside properties can be boolean values which state whether the sub-sentence can be derived from some non-terminal symbol of the grammar, or numeric values corresponding to the weight of (all or best) such derivations [17,20-22].

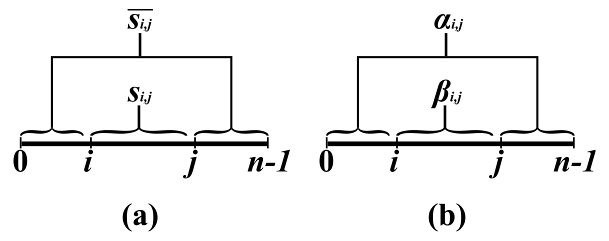


**Figure 2 Inside and outside sub-instances, and their corresponding properties**.

An *outside property* $\alpha_{i,j}$ is a property of the residual string obtained by removing $s_{i,j}$ from $s$ (i.e. the pair of strings $s_{0,i}$ and $s_{j,n}$, see Figure 2). Such a residual string is denoted by $\overline{s_{i,j}}$. Outside property computations occur in algorithms for the RNA Base Pair Binding Probabilities problem [10], and in the Inside-Outside algorithm for learning derivation rule weights for WCFGs [43].

In the rest of this paper, given an instance string $s$, substrings of the form $s_{i,j}$ and residual strings of the form $\overline{s_{i,j}}$ will be considered as *sub-instances* of $s$. Characters and positions in such sub-instances are indexed according to the same indexing as of the original string $s$. That is, the characters in sub-instances of the form $s_{i,j}$ are indexed from $i$ to $j - 1$, and in sub-instances of the form $\overline{s_{i,j}}$ the first $i$ characters are indexed between 0 and $i - 1$, and the remaining characters are indexed between $j$ and $n - 1$. The notation $\beta$ will be used to denote the set of all values of the form $\beta_{i,j}$ with respect to substrings $s_{i,j}$ of some given string $s$. It is convenient to visualize $\beta$ as an $||s|| \times ||s||$ matrix, where the $(i, j)$-th entry in the matrix contains the value $\beta_{i,j}$. Only entries in the upper triangle of the matrix $\beta$ correspond to valid substrings of $s$. For convenience, we define that values of the form $\beta_{i,j}$, when $j < i$, equal to $\varphi$ (with respect to the corresponding domain of values). Notations such as $\beta_{I,J}$, $\beta_{i,J}$, and $\beta_{I,j}$ are used in order to denote the corresponding sub-matrices of $\beta$, as defined above. Similar notations are used for a set $\alpha$ of outside properties.

## 3 The Inside Vector Multiplication Template

In this section we describe a template that defines a class of problems, called the *Inside Vector Multiplication Template* (*Inside VMT*). We start by giving a simple motivating example in Section 3.1. Then, the class of Inside VMT problems is formally defined in Section 3.2, and in Section 3.3 an efficient generic algorithm for all Inside VMT problems is presented.

## 3.1 Example: RNA Base-Pairing Maximization

The *RNA Base-Pairing Maximization* problem [6] is a simple variant of the *RNA Folding* problem, and it exhibits the main characteristics of Inside VMT problems. In this problem, an input string $s = s_0 s_1 \dots s_{n-1}$ represents a string of *bases* (or *nucleotides*) over the alphabet $A$, $C$, $G$, $U$. Besides strong (covalent) chemical bonds which occur between each pair of consecutive bases in the string, bases at distant positions tend to form additional weaker (hydrogen) bonds, where a base of type $A$ can pair with a base of type $U$, a base of type $C$ can pair with a base of type $G$, and in addition a base of type $G$ can pair with a base of type $U$. Two bases which can pair to each other in such a (weak) bond are called *complementary bases*, and a bond between two such bases is called a *base-pair*. The notation $a \bullet b$ is used to denote that the bases at indices $a$ and $b$ in $s$ are paired to each other.

A *folding* (or a *secondary structure*) of $s$ is a set $F$ of base-pairs of the form $a \bullet b$, where $0 \le a < b < n$, which sustains that there are no two distinct base pairs $a \bullet b$ and $c \bullet d$ in $F$ such that $a \le c \le b \le d$ (i.e. the paring is nested, see Figure 3). Denote by $|F|$ the number of complementary base-pairs in $F$. The goal of the RNA base-paring maximization problem is to compute the maximum number of complementary base-pairs in a folding of an input RNA string $s$. We call such a number the *solution* for $s$, and denote by $\beta_{i,j}$ the solution for the substring $s_{i,j}$. For substrings of the form $s_{i,i}$ and $s_{i,i+1}$ (i.e. empty strings or strings of length 1), the only possible folding is the empty folding, and thus $\beta_{i,i} = \beta_{i,i+1} = 0$. We next explain how to recursively compute $\beta_{i,j}$ when $j > i + 1$.

In order to compute values of the form $\beta_{i,j}$, we distinguish between two types of foldings for a substring $s_{i,j}$: foldings of type *I* are those which contain the base-pair $i \bullet (j - 1)$, and foldings of type *II* are those which do not contain $i \bullet (j - 1)$.

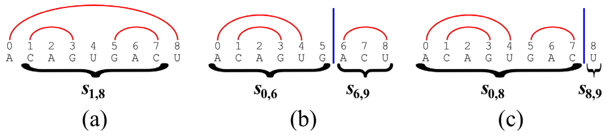


**Figure 3 An RNA string $s = s_{0,9}$ = *ACAGUGACU*, and three corresponding foldings**. (a) A folding of type *I*, obtained by adding the base-pair $i \bullet (j - 1) = 0 \bullet 8$ to a folding for $s_{i+1,j-1} = s_{1,8}$. (b) A folding of type *II*, in which the last index 8 is paired to index 6. The folding is thus obtained by combining two independent foldings: one for the prefix $s_{0,6}$, and one for the suffix $s_{6,9}$. (c) A folding of type *II*, in which the last index 8 is unpaired. The folding is thus obtained by combining a folding for the prefix $s_{0,8}$, and an empty folding for the suffix $s_{8,9}$.

Consider a folding $F$ of type *I*. Since $i \bullet (j - 1) \in F$, the folding $F$ is obtained by adding the base-pair $i \bullet (j - 1)$ to some folding $F'$ for the substring $s_{i+1,j-1}$ (Figure 3a). The number of complementary base-pairs in $F$ is thus $|F'| + 1$ if the bases $s_i$ and $s_{j-1}$ are complementary, and otherwise it is $|F'|$. Clearly, the number of complementary base-pairs in $F$ is maximized when choosing $F'$ such that $|F'| = \beta_{i+1,j-1}$. Now, Consider a folding $F$ of type *II*. In this case, there must exist some position $q \in (i, j)$, such that no base-pair $a \bullet b$ in $F$ sustains that $a < q \le b$. This observation is true, since if $j - 1$ is paired to some index $p$ (where $i < p < j - 1$), then $q = p$ sustains the requirement (Figure 3b), and otherwise $q = j - 1$ sustains the requirement (Figure 3c). Therefore, $q$ splits $F$ into two independent foldings: a folding $F'$ for the prefix $s_{i,q}$, and a folding $F''$ for the suffix $s_{q,j}$, where $|F| = |F'| + |F''|$. For a specific split position $q$, the maximum number of complementary base-pairs in a folding of type *II* for $s_{i,j}$ is then given by $\beta_{i,q} + \beta_{q,j}$, and taking the maximum over all possible positions $q \in (i, j)$ guarantees that the best solution of this form is found.

Thus, $\beta_{i,j}$ can be recursively computed according to the following formula:

$$\beta_{i,j} = \max \left\{ \begin{array}{ll} (I) & \beta_{i+1,j-1} + \delta_{i,j-1}, \\ (II) & \max\limits_{q \in (i,j)} \{\beta_{i,q} + \beta_{q,j}\} \end{array} \right\},$$

where $\delta_{i,j-1} = 1$ if $s_i$ and $s_{j-1}$ are complementary bases, and otherwise $\delta_{i,j-1} = 0$.

### 3.1.1 The classical algorithm

The recursive computation above can be efficiently implemented using dynamic programming (DP). For an input string $s$ of length $n$, the DP algorithm maintains the upper triangle of an $||s|| \times ||s||$ matrix $B$, where each entry $B_{i,j}$ in $B$ corresponds to a solution $\beta_{i,j}$. The entries in $B$ are filled, starting from short base-case entries of the form $B_{i,i}$ and $B_{i,i+1}$, and continuing by computing entries corresponding to substrings with increasing lengths. In order to compute a value $\beta_{i,j}$ according to the recurrence formula, the algorithm needs to examine only values of the form $\beta_{i',j'}$ such that $s_{i',j'}$ is a strict substring of $s_{i,j}$ (Figure 4). Due to the bottom-up computation, these values are already computed and stored in $B$, and thus each such value can be obtained in $\Theta(1)$ time.

Upon computing a value $\beta_{i,j}$, the algorithm needs to compute term (*II*) of the recurrence. This computation is of the form of a *vector multiplication operation* $\oplus_{q \in (i,j)} (\beta_{i,q} \otimes \beta_{q,j})$, where the multiplication variant is the *Max Plus* multiplication. Since all relevant values in $B$ are computed, this computation can be implemented by computing $B_{i,(i,j)} \otimes B_{(i,j),j}$ (the multiplication of the two darkened vectors in Figure 4), which takes $\Theta(j - i)$ running time. After computing term (*II*), the algorithm

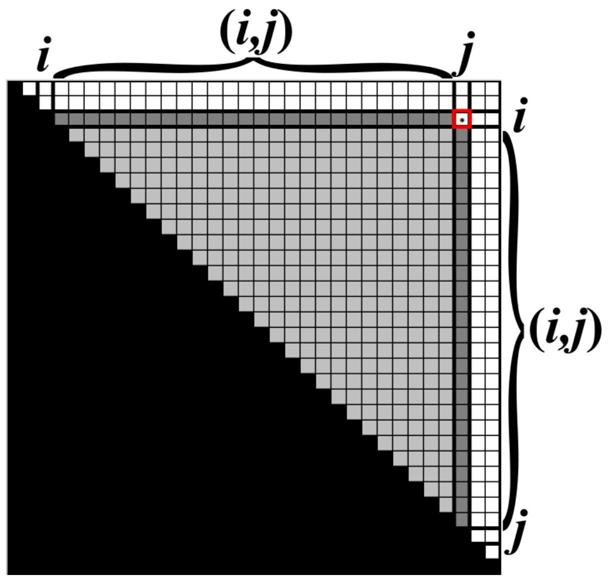


**Figure 4 The table *B* maintained by the DP algorithm**. In order to compute $B_{i,\ j}$, the algorithm needs to examine only values in entries of $B$ that correspond to strict substrings of $s_{i,\ j}$ (depicted as light and dark grayed entries).

needs to perform additional operations for computing $\beta_{i,\ j}$ which take $\Theta(1)$ running time (computing term (*I*), and taking the maximum between the results of the two terms). It can easily be shown that, on average, the running time for computing each value $\beta_{i,\ j}$ is $\Theta(n)$, and thus the overall running time for computing all $\Theta(n^2)$ values $\beta_{i,\ j}$ is $\Theta(n^3)$. Upon termination, the computed matrix $B$ equals to the matrix $\beta$, and the required result $\beta_{0,n}$ is found in the entry $B_{0,n}$.

### 3.2 Inside VMT definition

In this section we characterize the class of *Inside VMT* problems. The *RNA Base-Paring Maximization* problem, which was described in the previous section, exhibits a simple special case of an Inside VMT problem, in which the goal is to compute a single inside property for a given input string. Note that this requires the computation of such inside properties for all substrings of the input, due to the recursive nature of the computation. In other Inside VMT problems the case is similar, hence we will assume that the goal of Inside VMT problems is to compute inside properties for *all* substrings of the input string. In the more general case, an Inside VMT problem defines several inside properties, and all of these properties are computed for each substring of the input in a mutually recursive manner. Examples of such problems are the *RNA Partition Function* problem [10] (which is described in Appendix A), the *RNA Energy Minimization* problem [7] (which computes several folding scores for each substring of the input, corresponding to restricted types of foldings), and the *CFG Parsing* problem [20-22] (which computes, for every non-terminal symbol in the grammar and every sub-sentence of the input, a boolean value that indicates whether the sub-sentence can be derived in the grammar when starting the derivation from the non-terminal symbol).

A common characteristic of all Inside VMT problems is that the computation of at least one type of an inside property requires a result of a vector multiplication operation, which is of similar structure to the multiplication described in the previous section for the RNA Base-Paring Maximization problem. In many occasions, it is also required to output a *solution* that corresponds to the computed property, e.g. a minimum energy secondary structure in the case of the RNA folding problem, or a maximum weight parse-tree in the case of the WCFG Parsing problem. These solutions can usually be obtained by applying a traceback procedure over the computed dynamic programming tables. As the running times of these traceback procedures are typically negligible with respect to the time needed for filling the values in the tables, we disregard this phase of the computation in the rest of the paper.

The following definition describes the family of *Inside VMT problems*, which share common combinatorial characteristics and may be solved by a generic algorithm which is presented in Section 3.3.

**Definition 1** *A problem is considered an* Inside VMT problem *if it fulfills the following requirements.*

1. *Instances of the problem are strings, and the goal of the problem is to compute for every substring $s_{i,\ j}$ of an input string s, a series of inside properties $\beta^1_{i,j}, \beta^2_{i,j}, \ldots, \beta^K_{i,j}$.*
2. *Let $s_{i,\ j}$ be a substring of some input string s. Let $1 \le k \le K$, and let $\mu^k_{i,j}$ be a result of a vector multiplication of the form $\mu^k_{i,j} = \oplus_{q\in(i,j)}\left(\beta^{k'}_{i,q} \otimes \beta^{k''}_{q,j}\right)$, for some $1 \le k', k'' \le K$. Assume that the following values are available: $\mu^k_{i,j}$, all values $\beta^{k'}_{i',j'}$ for $1 \le k' \le K$ and $s_{i',j'}$ a strict substring of $s_{i,\ j}$,* and all values *$\beta^{k'}_{i,j}$ for $1 \le k' < k$. Then, $\beta^k_{i,j}$ can be computed in $o(||s||)$ running time.*
3. *In the multiplication variant that is used for computing $\mu^k_{i,j}$, the $\oplus$ operation is associative, and the domain of elements contains a zero element. In addition, there is a matrix multiplication algorithm for this multiplication variant, whose running time $M(n)$ over two $n \times n$ matrices satisfies $M(n) = o(n^3)$.*

Intuitively, $\mu^k_{i,j}$ reflects an expression which examines all possible splits of $s_{i,\ j}$ into a prefix substring $s_{i,\ q}$ and a suffix substring $s_{q,\ j}$ (Figure 5). Each split corresponds to

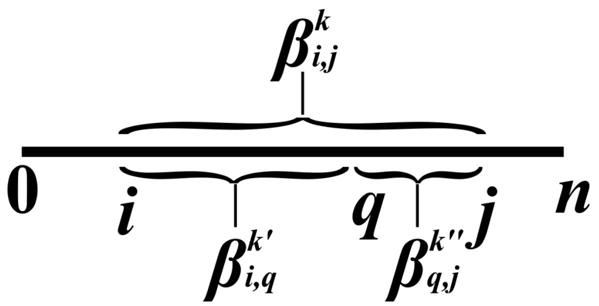


**Figure 5 The examination of a split position $q$ in the computation of an inside property $\beta^{k}_{i,j}$.**

a term that should be considered when computing the property $\beta^{k}_{i,j}$, where this term is defined to be the application of the $\otimes$ operator between the property $\beta^{k'}_{i,q}$ of the prefix $s_{i,\ q}$, and the property $\beta^{k''}_{q,j}$ of the suffix $s_{q,\ j}$ (where $\otimes$ usually stands for +, ·, or $\wedge$). The combined value $\mu^{k}_{i,j}$ for all possible splits is then defined by applying the $\oplus$ operation (usually min/max, +, or $\vee$) over these terms, in a sequential manner. The template allows examining $\mu^{k}_{i,j}$, as well as additional values of the form $\beta^{k'}_{i',j'}$, for strict substrings $s_{i',j'}$ of $s_{i,\ j}$ and $1 \leq k' < K$, and values of the form $\beta^{k'}_{i,j}$ for $1 \leq k' < k$, in order to compute $\beta^{k}_{i,j}$. In typical VMT problems (such as the RNA Base-Paring Maximization problem, and excluding problems which are described in Section 5), the algorithm needs to perform $\Theta(1)$ operations for computing $\beta^{k}_{i,j}$, assuming that $\mu^{k}_{i,j}$ and all other required values are pre-computed. Nevertheless, the requirement stated in the template is less strict, and it is only assumed that this computation can be executed in a sub-linear time with respect to $||s||$.

### 3.3 The Inside VMT algorithm

We next describe a generic algorithm, based on Valiant's algorithm [23], for solving problems sustaining the Inside VMT requirements. For simplicity, it is assumed that a single property $\beta_{i,\ j}$ needs to be computed for each substring $s_{i,\ j}$ of the input string $s$. We later explain how to extend the presented algorithm to the more general cases of computing $K$ inside properties for each substring.

The new algorithm also maintains a matrix $B$ as defined in Section 3.1. It is a divide-and-conquer recursive algorithm, where at each recursive call the algorithm computes the values in a sub-matrix $B_{I,\ J}$ of $B$ (Figure 6). The actual computation of values of the form $\beta_{i,\ j}$ is conducted at the base-cases of the recurrence, where the corresponding sub-matrix contains a single entry $B_{i,\ j}$. The main idea is that upon reaching this stage the term $\mu_{i,\ j}$ was already computed, and thus $\beta_{i,\ j}$ can be computed efficiently, as implied by item 2 of Definition 1. The accelerated computation of values of the form $\mu_{i,\ j}$ is obtained by the application of fast matrix multiplication subroutines between sibling recursive calls of the algorithm. We now turn to describe this process in more details.

At each stage of the run, each entry $B_{i,\ j}$ either contains the value $\beta_{i,j}$, or some intermediate result in the computation of $\mu_{i,\ j}$. Note that only the upper triangle of $B$ corresponds to valid substrings of the input. Nevertheless, our formulation handles all entries uniformly, implicitly ignoring values in entries $B_{i,\ j}$ when $j < i$. The following pre-condition is maintained at the beginning of the recursive call for computing $B_{I,\ J}$ (Figure 7):

1. Each entry $B_{i,\ j}$ in $B_{[I,J],\ [I,J]}$ contains the value $\beta_{i,\ j}$, except for entries in $B_{I,\ J}$.
2. Each entry $B_{i,\ j}$ in $B_{I,\ J}$ contains the value $\oplus_{q \in (I,J)} (\beta_{i,\ q} \otimes \beta_{q,\ j})$. In other words, $B_{I,\ J} = \beta_{I,(I,J)} \otimes \beta_{(I,J),J}$.

Let $n$ denote the length of $s$. Upon initialization, $I = J = [0, n]$, and all values in $B$ are set to $\varphi$. At this stage $(I, J)$ is an empty interval, and so the pre-condition with respect to the complete matrix $B$ is met. Now, consider a call to the algorithm with some pair of intervals $I$, $J$. If $I = [i, i]$ and $J = [j, j]$, then from the pre-condition, all values $\beta_{i',\ j'}$ which are required for the computation $\beta_{i,\ j}$ of are computed and stored in $B$, and $B_{i,\ j} = \mu_{i,\ j}$ (Figure 4). Thus, according to the Inside VMT requirements, $\beta_{i,\ j}$ can be evaluated in $o(||s||)$ running time, and be stored in $B_{i,\ j}$.

Else, either $|I| > 1$ or $|J| > 1$ (or both), and the algorithm partitions $B_{I,\ J}$ into two sub-matrices of approximately equal sizes, and computes each part recursively. This partition is described next. In the case where $|I| \leq |J|$, $B_{I,\ J}$ is partitioned vertically (Figure 8). Let $J_1$ and $J_2$ be two column intervals such that $J = J_1J_2$ and $|J_1| = \lfloor |J|/2 \rfloor$ (Figure 8b). Since $J$ and $J_1$ start at the same index, $(I, J) = (I, J_1)$. Thus, from the pre-condition and Equation 2, $B_{I,J_1} = \beta_{I,(I,J_1)} \otimes \beta_{(I,J_1),J_1}$. Therefore, the pre-condition with respect to the sub-matrix $B_{I,J_1}$ is met, and the algorithm computes this sub-matrix recursively. After $B_{I,J_1}$ is computed, the first part of the pre-condition with respect to $B_{I,J_2}$ is met, i.e. all necessary values for computing values in $B_{I,J_2}$, except for those in $B_{I,J_2}$ itself, are computed and stored in $B$. In addition, at this stage $B_{I,J_2} = \beta_{I,(I,J)} \otimes \beta_{(I,J),J_2}$. Let $L$ be the interval such that $(I, J_2) = (I, J)L$. $L$ is contained in $J_1$, where it can be verified that either $L = J_1$ (if the last index in $I$ is smaller than the first index in $J$, as in the example of Figure 8c), or $L$ is an empty interval (in all other cases which occur along the recurrence). To meet the full pre-condition requirements with respect to $I$ and

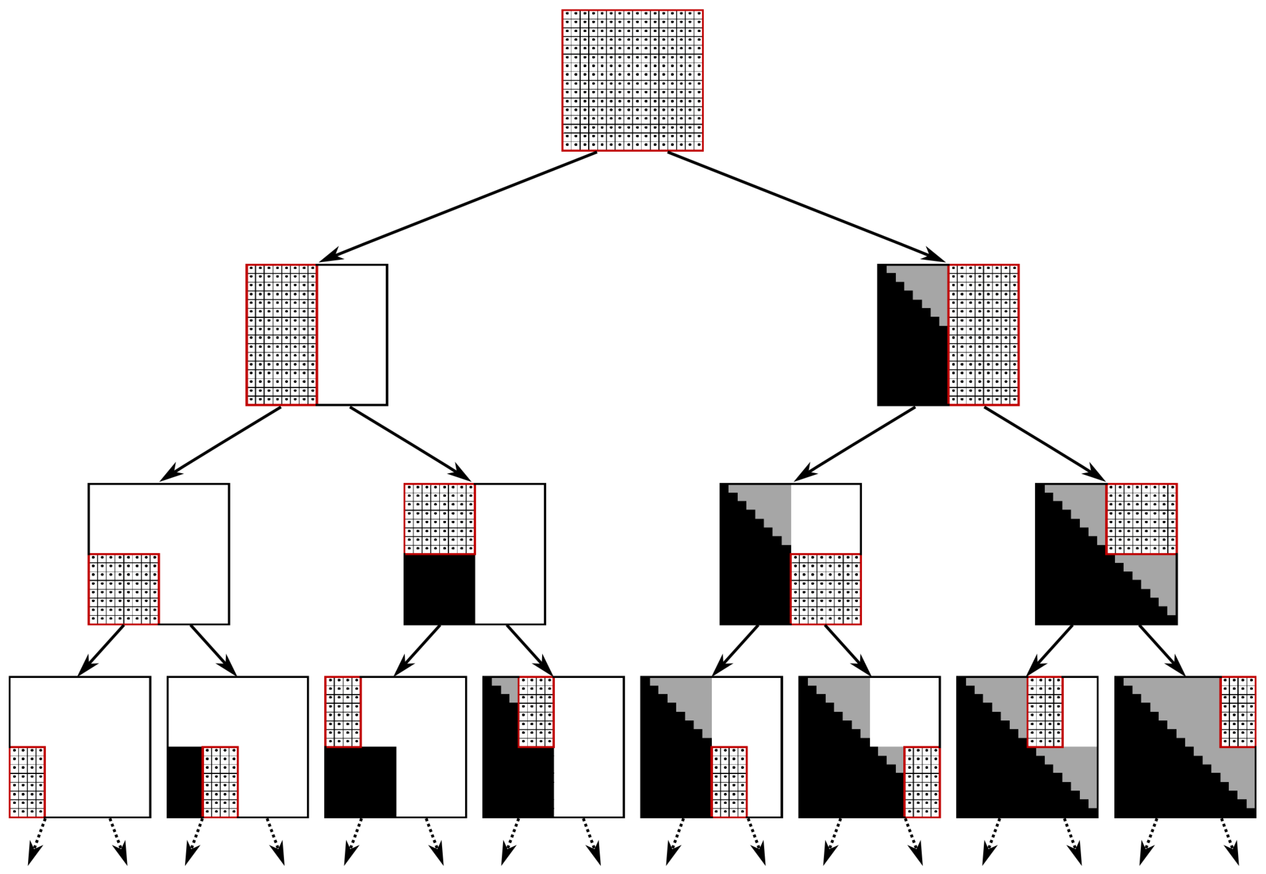


**Figure 6 The recursion tree**. Each node in the tree shows the state of the matrix *B* when the respective call to *Compute-Inside-Sub-Matrix* starts. The dotted cells are those that are computed during the call. Black and gray cells are cells whose values were already computed (black cells correspond to empty substrings). The algorithm starts by calling the recursive procedure over the complete matrix. Each visited sub-matrix is decomposed into two halves, which are computed recursively. The recursion visits the sub-matrices according to a pre-order scan on the tree depicted in the figure. Once the first among a pair of sibling recursive calls was concluded, the algorithm uses the new computed portion of data as an input to a fast matrix multiplication subroutine, which facilitate the computation of the second sibling.

$J_2$, $B_{I,J_2}$ is updated using Equation 3 to be $B_{I,J_2} \oplus (B_{I,L} \otimes B_{L,J_2}) = (\beta_{I,(I,J)} \otimes \beta_{(I,J),J_2}) \oplus (\beta_{I,L} \otimes \beta_{L,J_2}) = \beta_{I,(I,J_2)} \otimes \beta_{(I,J_2),J_2}$.

Now, the pre-condition with respect to $B_{I,J_2}$ is established, and the algorithm computes $B_{I,J_2}$ recursively. In the case where $|I| > |J|$, $B_{I,\,J}$ is partitioned horizontally, in a symmetric manner to the vertical partition. The horizontal partition is depicted in Figure 9. The complete pseudo-code for the Inside VMT algorithm is given in Table 2.

### *3.3.1 Time complexity analysis for the Inside VMT algorithm*

In order to analyze the running time of the presented algorithm, we count separately the time needed for computing the base-cases of the recurrence, and the time for non-base-cases.

In the base-cases of the recurrence (lines 1-2 in Procedure Compute-Inside-Sub-Matrix, Table 2), $|I| = |J| = 1$, and the algorithm specifically computes a value of the form $\beta_{i,j}$. According to the VMT requirements, each such value is computed in $o(||s||)$ running time. Since there are $\Theta(||s||^2)$ such base-cases, the overall running time for their computation is $o(||s||^3)$.

Next, we analyze the time needed for all other parts of the algorithm except for those dealing with the base-cases. For simplicity, assume that $||s|| = 2^x$ for some integer $x$. Then, due to the fact that at the beginning $|I| = |J| = 2^x$, it is easy to see that the recurrence encounters pairs of intervals $I, J$ such that either $|I| = |J|$ or $|I| = 2|J|$.

Denote by $T(r)$ and $D(r)$ the time it takes to compute all recursive calls (except for the base-cases) initiated from a call in which $|I| = |J| = r$ (exemplified in Figure 8) and $|I| = 2|J| = r$ (exemplified in Figure 9), respectively.

When $|I| = |J| = r$ (lines 4-9 in Procedure Compute-Inside-Sub-Matrix, Table 2), the algorithm performs two recursive calls with sub-matrices of size $r \times \frac{r}{2}$, a matrix multiplication between an $r \times \frac{r}{2}$ and an $\frac{r}{2} \times \frac{r}{2}$ matrices, and a matrix addition of two $r \times \frac{r}{2}$ matrices. Since the

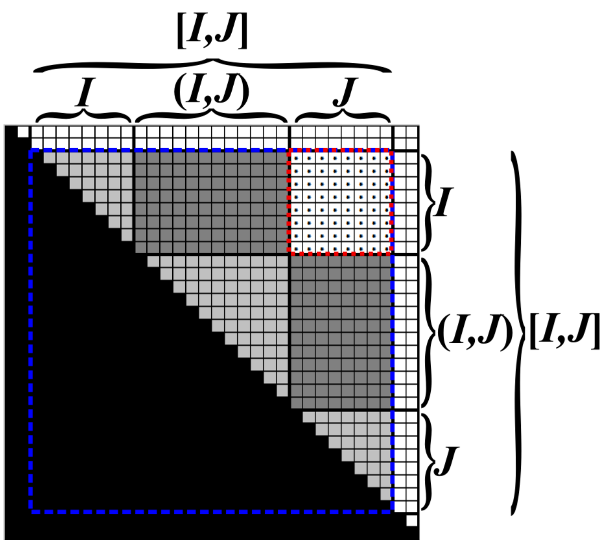


**Figure 7 The pre-condition for computing $B_{I,J}$ with the Inside VMT algorithm**. All values in $B_{[I,J],[I,J]}$, excluding $B_{I,J}$, are computed (light and dark grayed entries), and $B_{I,J} = \beta_{I,(I,J)} \otimes \beta_{(I,J),J} = B_{I,(I,J)} \otimes B_{(I,J),J}$ (the entries of $B_{I,(I,J)}$ and $B_{(I,J),J}$ are dark grayed).

matrix multiplication can be implemented by performing two $\frac{r}{2} \times \frac{r}{2}$ matrix multiplications (Equation 1), $T(r)$ is given by

$$T(r) = 2D(r) + 2M\left(\frac{r}{2}\right) + \Theta(r^2).$$

When $|I| = 2|J| = r$ (lines 10-15 in Procedure Compute-Inside-Sub-Matrix, Table 2), the algorithm performs two recursive calls with sub-matrices of size $\frac{r}{2} \times \frac{r}{2}$, a matrix multiplication between two $\frac{r}{2} \times \frac{r}{2}$ matrices, and a matrix addition of two $\frac{r}{2} \times \frac{r}{2}$ matrices. Thus, $D(r)$ is given by

$$D(r) = 2T\left(\frac{r}{2}\right) + M\left(\frac{r}{2}\right) + \Theta(r^2).$$

Therefore, $T(r) = 4T(\frac{r}{2}) + 4M(\frac{r}{2}) + \Theta(r^2)$. By the *master theorem* [44], $T(r)$ is given by

- $T(r) = \Theta(r^2 \log^{k+1} r)$, if $M(r) = O(r^2 \log^k r)$ for some $k \geq 0$, and
- $T(r) = \Theta(M(\frac{r}{2}))$, if $M(\frac{r}{2}) = \Omega(r^{2+\varepsilon})$ for some $\varepsilon > 0$, and $4M(\frac{r}{2}) \leq dM(r)$ for some constant $d < 1$ and sufficiently large $r$.

The running time of all operations except for the computations of base cases is thus $T(||s||)$. In both cases listed above, $T(||s||) = o(||s||^3)$, and therefore the overall running time of the algorithm is sub-cubic running time with respect to the length of the input string.

The currently best algorithms for the three standard multiplication variants described in Section 2.2 satisfy that $M(r) = \Omega(r^{2+\varepsilon})$, and imply that $T(r) = \Theta(M(r))$. When this case holds, and the time complexity of computing the base-cases of the recurrence does not exceed $M(||s||)$ (i.e. when the amortized running time for computing each one of the $\Theta(||s||^2)$ base cases is $O\left(\frac{M(||s||)}{||s||^2}\right)$), we say that the problem sustains the *standard VMT settings*. The running time of the VMT algorithm over such problems is thus $\Theta(M(||s||))$. All realistic inside VMT problems familiar to the authors sustain the standard VMT settings.

### 3.3.2 Extension to the case where several inside properties are computed

We next describe the modification to the algorithm for the general case where the goal is to compute a series of inside property-sets $\beta^1, \beta^2, ..., \beta^K$ (see Appendix A for an

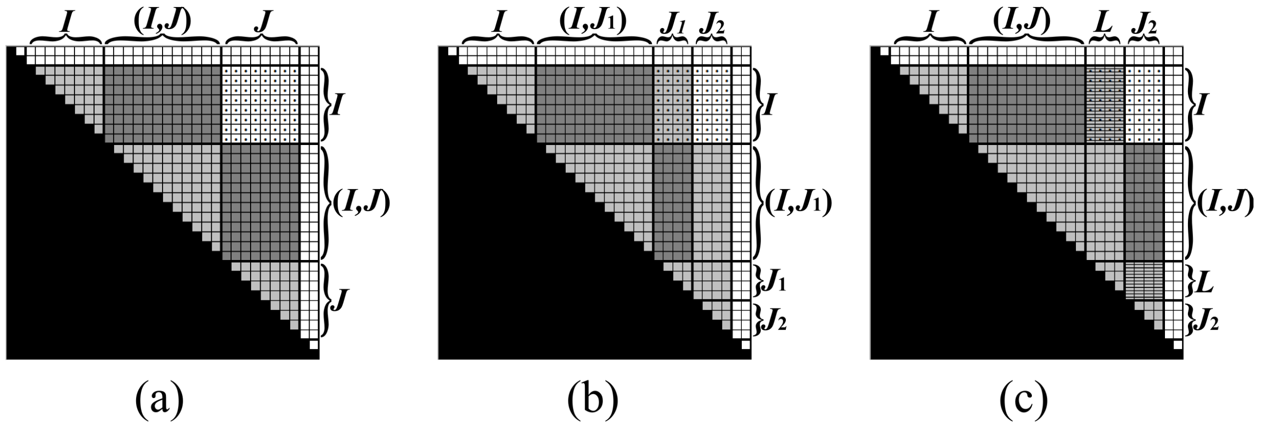


**Figure 8 An exemplification of the vertical partition of $B_{I,J}$ (the entries of $B_{I,J}$ are dotted)**. (a) The pre-condition requires that all values in $B_{[I,J],[I,J]}$, excluding $B_{I,J}$, are computed, and $B_{I,J} = \beta_{I,(I,J)} \otimes \beta_{(I,J),J}$ (see Figure 7). (b) $B_{I,J}$ is partitioned vertically to $B_{I,J_1}$ and $B_{I,J_2}$, where $B_{I,J_1}$ is computed recursively. (c) The pre-condition for computing $B_{I,J_2}$ is established, by updating $B_{I,J_2}$ to be $B_{I,J_2} \oplus (B_{I,L} \otimes B_{L,J_2})$ (in this example $L = J_1$, since $I$ ends before $J_1$ starts). Then, $B_{I,J_2}$ is computed recursively (not shown).

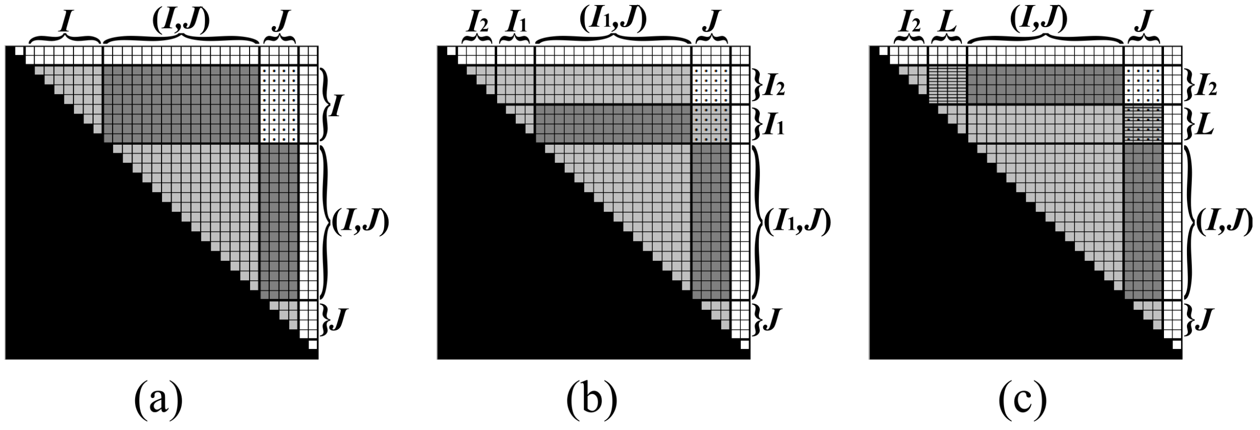


**Figure 9 An exemplification of the horizontal partition of $B_{I,J}$.** See Figure 8 for the symmetric description of the stages.

example of such a problem). The algorithm maintains a series of matrices $B^1, B^2, ..., B^K$, where $B^k$ corresponds to the inside property-set $\beta^k$. Each recursive call to the algorithm with a pair of intervals $I$, $J$ computes the series of sub-matrices $B^1_{I,J}, B^2_{I,J}, \ldots, B^K_{I,J}$. The pre-condition at each stage is:

1. For all $1 \leq k \leq K$, all values in $B^k_{[I,J],[I,J]}$ are computed, excluding the values in $B^k_{I,J}$,
2. If a result of a vector multiplication of the form $\mu^k_{i,j} = \oplus_{q\in(i,j)}\left(\beta^{k'}_{i,q} \otimes \beta^{k''}_{q,j}\right)$ is required for the computation of $\beta^k_{i,j}$, then $B^k_{I,J} = \beta^{k'}_{I,(I,J)} \otimes \beta^{k''}_{(I,J),J}$.

**Table 2 The Inside VMT algorithm**

```
Inside-VMT (s)
1:  Allocate a matrix B of size ||s|| × ||s||, and initialize all entries in B with φ elements.
2:  Call Compute-Inside-Sub-Matrix ([0, n], [0, n]), where n is the length of s.
3:  return B

Compute-Inside-Sub-Matrix (I, J)
pre-condition: The values in $B_{[I,J],[I,J]}$, excluding the values in $B_{I,J}$, are computed, and $B_{I,J} = \beta_{I,(I,J)} \otimes \beta_{(I,J),J}$.
post-condition: $B_{[I,J],[I,J]} = \beta_{[I,J],[I,J]}$.
1:  if I = [i, i] and J = [j, j] then
2:    If $i \leq j$, compute $\beta_{i,j}$ (in $o(||s||)$ running time) by querying computed values in B and the value $\mu_{i,j}$ which is stored in $B_{i,j}$. Update $B_{i,j} \leftarrow \beta_{i,j}$.
3:  else
4:    if $|I| \leq |J|$ then
5:      Let $J_1$ and $J_2$ be the two intervals such that $J_1J_2 = J$, and $|J_1| = \lfloor |J|/2 \rfloor$.
6:      Call Compute-Inside-Sub-Matrix ($I, J_1$).
7:      Let L be the interval such that $(I, J)L = (I, J_2)$.
8:      Update $B_{I,J_2} \leftarrow B_{I,J_2} \oplus (B_{I,L} \otimes B_{L,J_2})$.
9:      Call Compute-Inside-Sub-Matrix ($I, J_2$).
10:   else
11:     Let $I_1$ and $I_2$ be the two intervals such that $I_2I_1 = I$, and $|I_2| = \lfloor |I|/2 \rfloor$.
12:     Call Compute-Inside-Sub-Matrix ($I_1, J$).
13:     Let L be the interval such that $L(I, J) = (I_2, J)$.
14:     Update $B_{I_2,J} \leftarrow (B_{I_2,L} \otimes B_{L,J}) \oplus B_{I_2,J}$.
15:     Call Compute-Inside-Sub-Matrix ($I_2, J$).
16:   end if
17: end if
```

The algorithm presented in this section extends to handling this case in a natural way, where the modification is that now the matrix multiplications may occur between sub-matrices taken from different matrices, rather than from a single matrix. The only delicate aspect here is that for the base case of the recurrence, when $I = [i, i]$ and $J = [j, j]$, the algorithm needs to compute the values in the corresponding entries in a sequential order $B^1_{i,j}, B^2_{i,j}, \ldots, B^K_{i,j}$, since it is possible that the computation of a property $\beta^k_{i,j}$ requires the availability of a value of the form $\beta^{k'}_{i,j}$ for some $k' < k$. Since $K$ is a constant which is independent of the length on the input string, it is clear that the running time for this extension remains the same as for the case of a single inside value-set.

The following Theorem summarizes our main results with respect to Inside VMT problems.

**Theorem 1** *For every Inside VMT problem there is an algorithm whose running time over an instance s is* $o(||s||^3)$. *When the problem sustains the standard VMT settings, the running time of the algorithm is* $\Theta(M(||s||))$, *where* $M(n)$ *is the running time of the corresponding matrix multiplication algorithm over two* $n \times n$ *matrices.*

## 4 Outside VMT

In this section we discuss how to solve another class of problems, denoted *Outside VMT* problems, by modifying the algorithm presented in the previous section. Similarly to Inside VMT problems, the goal of Outside VMT problems is to compute sets of outside properties $\alpha^1, \alpha^2, ..., \alpha^K$ corresponding to some input string (see notations in Section 2.3).

Examples for problems which require outside properties computation and adhere to the VMT requirements are the *RNA Base Pair Binding Probabilities* problem [10] (described in Appendix A) and the *WCFG Inside-Outside* problem [43]. In both problems, the computation of outside properties requires a set of pre-computed inside properties, where these inside properties can be computed with the Inside VMT algorithm. In such cases, we call the problems *Inside-Outside VMT* problems.

The following definition describes the family of *Outside VMT* problems.

**Definition 2** *A problem is considered an* Outside VMT *problem if it fulfills the following requirements.*

1. *Instances of the problem are strings, and the goal of the problem is to compute for every sub-instance* $\overline{s_{i,j}}$ *of an input string s, a series of outside properties* $\alpha^1_{i,j}, \alpha^2_{i,j}, \ldots, \alpha^K_{i,j}$.
2. *Let* $\overline{s_{i,j}}$ *be a sub-instance of some input string s. Let* $1 \leq k \leq K$, *and let* $\mu^k_{i,j}$ *be a result of a vector multiplication of the form* $\mu^k_{i,j} = \oplus_{q\in[0,i)} \left(\beta^k_{q,i} \otimes \alpha^{k'}_{q,j}\right)$ *or of the form* $\mu^k_{i,j} = \oplus_{q\in(j,n]} \left(\alpha^{k'}_{i,q} \otimes \beta^k_{j,q}\right)$, *for some* $1 \leq k' \leq K$ *and a set of pre-computed inside properties* $\beta^k$. *Assume that the following values are available:* $\mu^k_{i,j}$, *all values* $\alpha^{k'}_{i',j'}$ *for* $1 \leq k' \leq K$ *and* $s_{i,j}$ *a strict substring of* $s_{i',j'}$, *and all values* $\alpha^{k'}_{i,j}$ *for* $1 \leq k' < k$. *Then,* $\alpha^k_{i,j}$ *can be computed in* $o(||s||)$ *running time.*
3. *In the multiplication variant that is used for computing* $\mu^k_{i,j}$, *the* $\oplus$ *operation is associative, and the domain of elements contains a zero element. In addition, there is a matrix multiplication algorithm for this multiplication variant, which running time* $M(n)$ *over two* $n \times n$ *matrices satisfies* $M(n) = o(n^3)$.

Here, for the case where $\mu^k_{i,j} = \oplus_{q\in[0,i)} \left(\beta^k_{q,i} \otimes \alpha^{k'}_{q,j}\right)$, the value $\mu^k_{i,j}$ reflects an expression which examines all possible splits of $\overline{s_{i,j}}$ into a sub-instance of the form $\overline{s_{q,j}}$, where $q < i$, and a sub-instance of the form $s_{q,i}$ (Figure 10a). Each split induces a term which is obtained by applying the $\otimes$ operation between a property $\beta^k_{q,i}$ of $s_{q,i}$ and a property $\alpha^{k'}_{q,j}$ of $\overline{s_{q,j}}$. Then, $\mu^k_{i,j}$ combines the values of all such terms by applying the $\oplus$ operator. Symmetrically, when $\mu^k_{i,j} = \oplus_{q\in(j,n]} \left(\alpha^{k'}_{i,q} \otimes \beta^k_{j,q}\right)$, $\mu^k_{i,j}$ reflects a value corresponding to the examination of all splits of $\overline{s_{i,j}}$ into a sub-instance of the form $\overline{s_{i,q}}$, where $q > j$, and a sub-instance of the form $s_{j,q}$ (Figure 10b).

We now turn to describe a generic recursive algorithm for Outside VMT problems. For simplicity, we consider the case of a problem whose goal is to compute a single set of outside properties $\alpha$, given a single pre-computed set of inside properties $\beta$. As for the Inside VMT algorithm, it is simple to extend the presented algorithm to the case where the goal is to compute a series of outside properties for every sub-instance of the input.

For an input string $s$ of length $n$, the algorithm maintains a matrix $A$ of size $||s|| \times ||s||$, corresponding to the required output matrix $\alpha$. In order to compute a property $\alpha_{i,j}$, the availability of values of the form $\alpha_{i',j'}$,

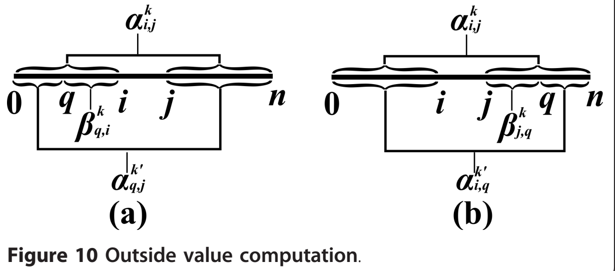


**Figure 10** Outside value computation.

such that $s_{i,\ j}$ is a strict substring of $s_{i',j'}$, is required. In terms of the matrix $A$, this means that when computing $A_{i,\ j}$, all entries in the sub-matrix $A_{[0,i],\ [j,n]}$, excluding the entry $A_{i,\ j}$ itself, need to be computed (Figure 11a).

At each recursive call, the algorithm computes the values in a sub-matrix of $A$. The following pre-condition is maintained when the algorithm is called over a sub-matrix $A_{I,\ J}$ (Figure 12):

1. Each entry $A_{i,\ j}$ in $A_{[0,I],\ [J,n]}$ contains the value $\alpha_{i,\ j}$, except for entries in $A_{I,\ J}$.
2. If the computation of $\alpha_{i,\ j}$ requires the result of a vector multiplication of the form $\mu_{i,\ j} = \oplus_{q\in [0,i)} (\beta_{q,\ i} \otimes \alpha_{q,\ j})$, then $A_{I,\ J} = (\beta_{[0,I),I})^T \otimes \alpha_{[0,I),J}$. Else, if the computation of $\alpha_{i,\ j}$ requires the result of a vector multiplication of the form $\mu_{i,\ j} = \oplus_{q\in (j,n]} (\alpha_{i,\ q} \otimes \beta_{j,\ q})$, then $A_{I,\ J} = \alpha_{I,\ (j,\ n]} \otimes (\beta_{j(J,\ n]})^T$.

The pre-condition implies that when $I = [i, i]$ and $J = [j, j]$, all necessary values for computing $\alpha_{i,\ j}$ in $o(||s||)$ running time are available, and thus $\alpha_{i,\ j}$ can be efficiently computed and stored in $A_{i,\ j}$. When $|I| >1$ or $|J| >1$, the algorithm follows a similar strategy as that of the Inside VMT algorithm, by partitioning the matrix into two parts, and computing each part recursively. The vertical and horizontal partitions are illustrated in Figure 13 and 14, respectively. The pseudo-code for the complete Outside VMT algorithm is given in Table 3. Similar running time analysis as that applied in the case of the Inside VMT algorithm can be shown, yielding the result stated in Theorem 2.

**Theorem 2** *For every Outside VMT problem there is an algorithm whose running time over an instance s is* $o(||s||^3)$. *When the problem sustains the standard VMT settings, the running time of the algorithm is* $\Theta(M(||s||))$, *where* $M(n)$ *is the running time of the corresponding matrix multiplication algorithm over two* $n \times n$ *matrices.*

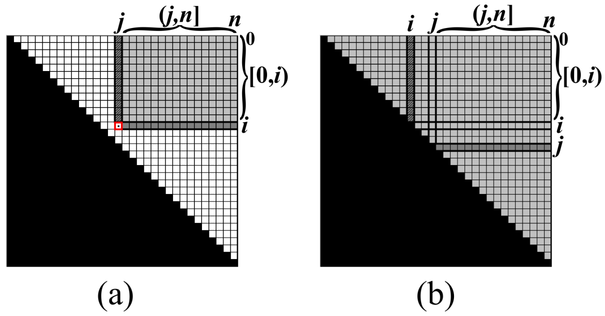


**Figure 11 The base case of the Outside VMT algorithm**. (a) An illustration of the matrix $A$ upon computing $A_{i,\ j}$. (b) An illustration of the pre-computed matrix $\beta$. All required values of the form $\alpha_{i',\ j'}$ for computing $\alpha_{i,\ j}$ have already been computed in the sub-matrix $A_{[0,i],\ [j,n]}$, excluding the entry $A_{i,\ j}$ itself. $\mu_{i,\ j}$ is obtained either by the multiplication $(\beta_{[0,i),i})^T \otimes A_{[0,i),\ j}$ (the multiplication of the transposed striped dark gray vector in $\beta$ with the striped dark gray vector in $A$), or by the multiplication $A_{i,(j,\ n]} \otimes (\beta_{j,(j,\ n]})^T$ (the multiplication of the non-striped dark gray vector in $A$ with the transposed non-striped dark gray vector in $\beta$).

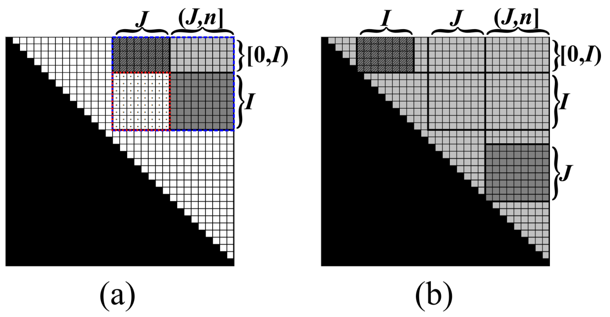


**Figure 12 The pre-condition of the Outside VMT algorithm**. (a) An illustration of the matrix $A$ upon computing $A_{I,\ J}$ (dotted entries). (b) An illustration of the pre-computed matrix $\beta$. The pre-condition requires that all entries in the sub-matrix $A_{[0,I],\ [J,n]}$, excluding $A_{I,\ J}$, are computed (dark and light gray entries). In addition, $A_{I,\ J}$ is either the result of the matrix multiplication $(\beta_{[0,I),I})^T \otimes \alpha_{[0,I),\ J}$ (the multiplication of the transposed striped dark gray matrix in $\beta$ with the striped dark gray matrix in $A$), or the multiplication $\alpha_{I,(J,\ n]} \otimes (\beta_{J,\ (J,\ n]})^T$ (the multiplication of the non-striped dark gray matrix in $A$ with the transposed non-striped dark gray matrix in $\beta$).

## 5 Multiple String VMT

In this section we describe another extension to the VMT framework, intended for problems for which the instance is a set of strings, rather than a single string. Examples of such problems are the *RNA Simultaneous Alignment and Folding* problem [15,37], which is described in details in Appendix B, and the *RNA-RNA Interaction* problem [9]. Additional problems which exhibit a slight divergence from the presented template, such as the *RNA-RNA Interaction Partition Function* problem [12] and the *RNA Sequence to Structured-Sequence Alignment* problem [13], can be solved in similar manners.

In order to define the Multiple String VMT variant in a general manner, we first give some related notation. An *instance* of a Multiple String VMT problem is a set of strings $S = (s^0, s^1, ..., s^{m-1})$, where the length of a string $s^p \in S$ is denoted by $n_p$. A *position* in $S$ is a set of indices $X = (i_0, i_1, ..., i_{m-1})$, where each index $i_p \in X$ is in the range $0 \le i_p \le n_p$. The number of different positions in $S$ is denoted by $||S|| = \prod_{0\le p<m} ||s^p||$.

Let $X = (i_0, i_1, ..., i_{m-1})$ and $Y = (j_0, j_1, ..., j_{m-1})$ be two positions in $S$. Say that $X \le Y$ if $i_p \le j_p$ for every $0 \le p < m$, and say that $X < Y$ if $X \le Y$ and $X \ne Y$. When $X \le Y$, denote by $S_{X,\ Y}$ the *sub-instance* $S_{X,Y} = \left(s^0_{i_0,j_0},\ s^1_{i_1,j_1},\ \dots,\ s^{m-1}_{i_{m-1},j_{m-1}}\right)$ of $S$ (see Figure 15a).

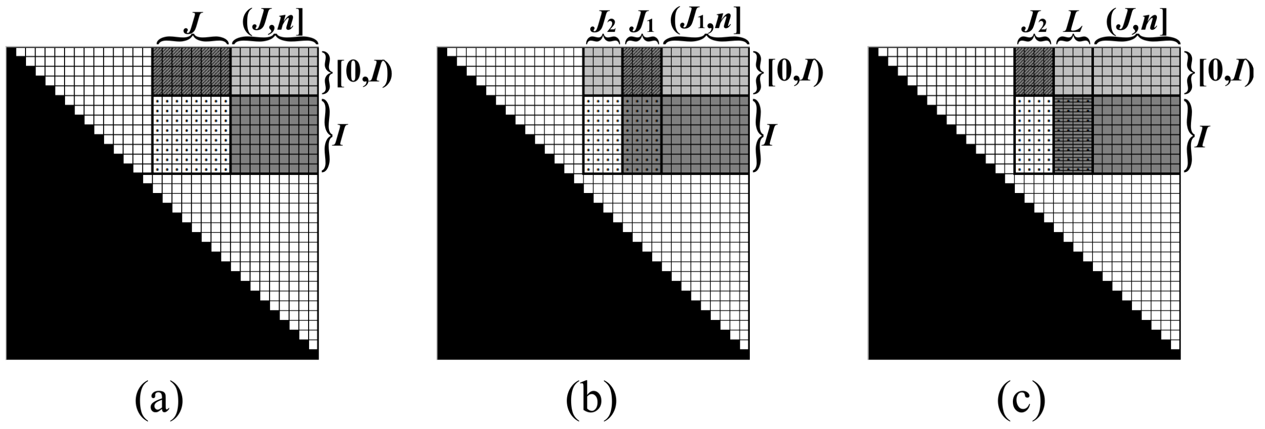


**Figure 13 The vertical decomposition in the Outside VMT algorithm, for the case where $|I| \le |J|$.** (a) $A_{I,J}$ sustains the pre-condition (see Figure 12 for notations). (b) $A_{I,J}$ is partitioned vertically to $A_{I,J_1}$ and $A_{I,J_2}$, where $A_{I,J_1}$ is computed recursively. (c) If $\mu_{i,j}$ is of the form $\oplus_{q\in(j,n]}(\alpha_{i,q} \otimes \beta_{j,q})$, then the pre-condition for computing $A_{I,J_2}$ is established, by updating $A_{I,J_2}$ to be $\left(A_{I,L} \otimes (\beta_{J_2,L})^T\right) \oplus A_{I,J_2}$ (in this example $L = J_1$, since $I$ ends before $J_1$ starts. The corresponding sub-matrix of $\beta$ is not shown). In the case where $\mu_{i,j} = \oplus_{q\in[0,i)}(\beta_{q,i} \otimes \alpha_{q,j})$, the pre-condition is met without any need for performing a matrix multiplication. Then, $A_{I,J_2}$ is computed recursively (not shown).

Say that $S_{X',Y'}$ is a *strict sub-instance* of $S_{X,Y}$ if $X \le X' \le Y' \le Y$, and $S_{X',Y'} \ne S_{X,Y}$.

The notation $X \nleq Y$ is used to indicate that it is not true that $X \le Y$. Here, the relation '≤' is not a linear relation, thus $X \nleq Y$ does not necessarily imply that $Y < X$. In the case where $X \nleq Y$, we say that $S_{X,Y}$ does not correspond to a *valid sub-instance* (Figure 15b). The notations $\bar{0}$ and $N$ will be used in order to denote the first position $(0, 0, ..., 0)$ and the last position $(n_0, n_1, ..., n_{m-1})$ in $S$, respectively. The notations which were used previously for intervals, are extended as follows: $[X, Y]$ denotes the *set* of all positions $Q$ such that $X \le Q \le Y$, $(X, Y)$ denotes the *set* of all positions $Q$ such that $X < Q < Y$, $[X, Y)$ denotes the *set* of all positions $Q$ such that $X \le Q < Y$, and $(X, Y]$ denotes the *set* of all positions $Q$ such that $X < Q \le Y$. Note that while previously these notations referred to intervals with sequential order defined over their elements, now the notations correspond to sets, where we assume that the order of elements in a set is unspecified.

Inside and outside properties with respect to multiple string instances are defined in a similar way as for a single string: An inside property $\beta_{X,Y}$ is a property that depends only on the sub-instance $S_{X,Y}$, where an outside property $\alpha_{X,Y}$ depends on the residual sub-instance of $S$, after excluding from each string in $S$ the

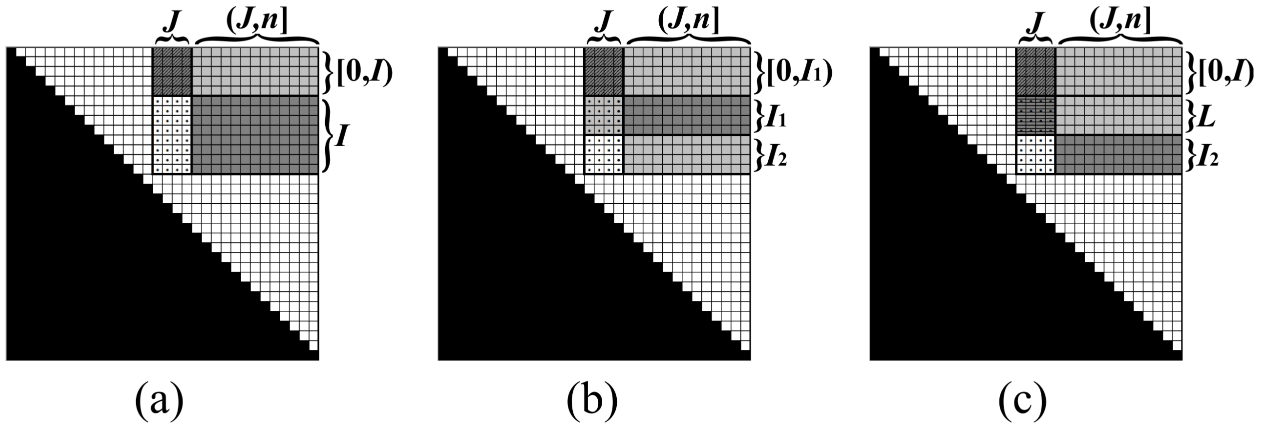


**Figure 14 The horizontal decomposition in the Outside VMT algorithm, for the case where $|I| > |J|$.** (a) $A_{I,J}$ sustains the pre-condition (see Figure 12 for notations). (b) $A_{I,J}$ is partitioned horizontally to $A_{I_1,J}$ and $A_{I_2,J}$, where $A_{I_1,J}$ is computed recursively. (c) If $\mu_{i,j}$ is of the form $\oplus_{q\in[0,i)}(\beta_{q,i} \otimes \alpha_{q,j})$, then the pre-condition for computing $A_{I_2,J}$ is established, by updating $A_{I_2,J}$ to be $A_{I_2,J} \oplus \left((\beta_{L,I_2})^T \otimes A_{L,J}\right)$ (in this example $L = I_1$, since $I_2$ ends before $J$ starts. The corresponding sub-matrix of $\beta$ is not shown). In the case where $\mu_{i,j} = \oplus_{q\in(j,n]}(\alpha_{i,q} \otimes \beta_{j,q})$, the pre-condition is met without any need for performing a matrix multiplication. Then, $A_{I_2,J}$ is computed recursively (not shown).

**Table 3 The outside VMT algorithm**

```
Outside-VMT ($s$, $\beta$)
1:  Allocate a matrix $A$ of size $||s|| \times ||s||$, and initialize all entries in $A$ with $\varphi$ elements.
2:  Call Compute-Outside-Sub-Matrix ([0, $n$], [0, $n$]), where $n$ is the length of $s$.
3:  return $A$

Compute-Outside-Sub-Matrix ($I$, $J$)
pre-condition: The values in $A_{[0,I], [J,n]}$, excluding the values in $A_{I,J}$, are computed.
  If $\mu_{i,j} = \oplus_{q\in[0,i)} (\beta_{q,i} \otimes \alpha_{q,j})$, then $A_{I,J} = (\beta_{[0,I),I})^T \otimes \alpha_{[0,I),J}$. Else, if $\mu_{i,j} = \oplus_{q\in(j,n]} (\alpha_{i,q} \otimes \beta_{j,q})$, then $A_{I,J} = \alpha_{I,(J,n]} \otimes (\beta_{J,(J,n]})^T$.
post-condition: $A_{[0,I], [J,n]} = \alpha_{[0,I], [J,n]}$.
1:  if $I = [i, i]$ and $J = [j, j]$ then
2:      If $i \le j$, compute $\alpha_{i,j}$ (in $o$ ($||s||$) running time) by querying computed values in $A$ and the value $\mu_{i,j}$ which is stored in $A_{i,j}$. Update $A_{i,j} \leftarrow \alpha_{i,j}$.
3:  else
4:      if $|I| \le |J|$ then
5:          Let $J_1$ and $J_2$ be the two intervals such that $J_2J_1 = J$, and $|J_2| = \lfloor |J|/2 \rfloor$.
6:          Call Compute-Outside-Sub-Matrix ($I$, $J_1$).
7:          if $\mu_{i,j}$ is of the form $\oplus_{q\in(j,n]} (\alpha_{i,q} \otimes \beta_{j,q})$ then
8:              Let $L$ be the interval such that $L(J, n] = (J_2, n]$.
9:              Update $A_{I,J_2} \leftarrow \left(A_{I,L} \otimes (\beta_{J_2,L})^T\right) \oplus A_{I,J_2}$.
10:         end if
11:         Call Compute-Outside-Sub-Matrix ($I$, $J_2$).
12:     else
13:         Let $I_1$ and $I_2$ be the two intervals such that $I_1I_2 = I$, and $|I_1| = \lfloor |I|/2 \rfloor$.
14:         Call Compute-Outside-Sub-Matrix ($I_1$, $J$).
15:         if $\mu_{i,j}$ is of the form $\oplus_{q\in[0,i)} (\beta_{q,i} \otimes \alpha_{q,j})$ then
16:             Let $L$ be the interval such that $[0, I)L = [0, I_2)$.
17:             Update $A_{I_2,J} \leftarrow A_{I_2,J} \oplus \left((\beta_{L,I_2})^T \otimes A_{L,J}\right)$.
18:         end if
19:         Call Compute-Outside-Sub-Matrix ($I_2$, $J$).
20:     end if
21: end if
```

corresponding substring in $S_{X,\ Y}$. In what follows, we define *Multiple String Inside VMT* problems, and show how to adopt the Inside VMT algorithm for such problems. The "outside" variant can be formulated and solved in a similar manner.

**Definition 3** *A problem is considered a* Multiple String Inside VMT *problem if it fulfills the following requirements.*

*1. Instances of the problem are sets of strings, and the goal of the problem is to compute for every sub-instance $S_{X,\ Y}$ of an input instance S, a series of inside properties $\beta^1_{X,Y}$, $\beta^2_{X,Y}$, ..., $\beta^K_{X,Y}$.*
*2. Let $S_{X,\ Y}$ be a sub-instance of some input instance S. Let $1 \le k \le K$, and let $\mu^k_{X,Y}$ be a value of the form $\mu^k_{X,Y} = \oplus_{Q\in(X,Y)}\left(\beta^{k'}_{X,Q} \otimes \beta^{k''}_{Q,Y}\right)$, for some $1 \le k', k'' \le K$. Assume that the following values are available: $\mu^k_{X,Y}$, all values $\beta^{k'}_{X',Y'}$ for $1 \le k' \le K$ and $S_{X',\ Y'}$ a strict sub-instance of $S_{X,\ Y}$, and all values $\beta^{k'}_{X,Y}$ for $1 \le k' < k$. Then, $\beta^k_{X,Y}$ can be computed in $o$ ($||S||$) running time.*
*3. In the multiplication variant that is used for computing $\mu^k_{X,Y}$, the $\oplus$ operation is associative and commutative, and the domain of elements contains a zero element. In addition, there is a matrix multiplication algorithm for this multiplication variant, which running time $M(n)$ over two $n \times n$ matrices satisfies $M(n) = o(n^3)$.*

Note that here there is an additional requirement with respect to the single string variant, that the $\oplus$ operator is *commutative*. This requirement was added, since while in the single string variant split positions in the interval $(i, j)$ could have been examined in a sequential order, and the (single string) Inside VMT algorithm retains this order when evaluating $\mu^k_{i,j}$, here there is no such natural sequential order defined over the positions in the set $(X, Y)$. The $\oplus$ commutativity requirement is met in all standard variants of matrix multiplication, and thus does not pose a significant restriction in practice.

Consider an instance $S = (s^0, s^1, ..., s^{m-1})$ for a Multiple String Inside VMT problem, and the simple case where a single property set $\beta$ needs to be computed (where $\beta$ corresponds to all inside properties of the form $\beta_{X,\ Y}$). Again, we compute the elements of $\beta$ in a square matrix

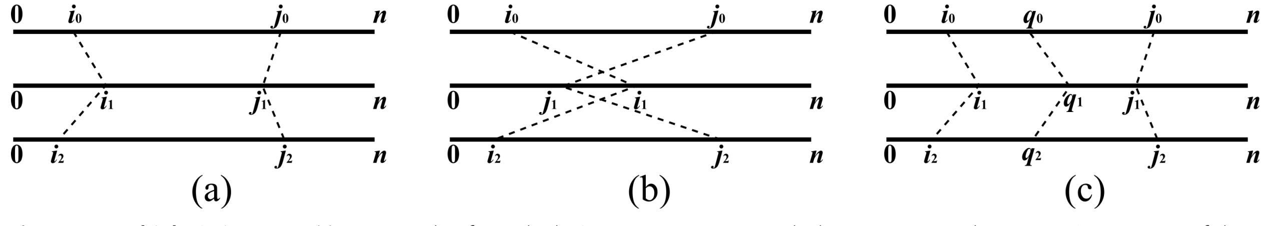


**Figure 15 Multiple String VMT**. (a) An example of a Multiple String VMT instance, with three strings. A sub-instance $S_{X,\ Y}$ consists of three substrings (where $X = \{i_0, i_1, i_2\}$ and $Y = \{j_0, j_1, j_2\}$). (b) Here, since $j_1 < i_1$ we have that $X \nleq Y$, and thus $S_{X,\ Y}$ does not correspond to a valid sub-instance. (c) A valid split of the sub-instance is obtained by splitting each one of the corresponding substrings $s^p_{i_p,j_p}$ into a prefix $s^p_{i_p,q_p}$ and a suffix $s^p_{q_p,j_p}$.

$B$ of size $||S|| \times ||S||$, and show that values of the form $\mu_{X,\ Y}$ correspond to results of vector multiplications within this matrix. For simplicity, assume that all string $s^p \in S$ are of the same length $n$, and thus $||S|| = (n + 1)^m$ (this assumption may be easily relaxed).

Define a one-to-one and onto mapping $h$ between positions in $S$ and indices in the interval $[0, ||S||)$, where for a position $X = (i_0, i_1, ..., i_{m-1})$ in $S$, $h(X) = \sum_{p=0}^{m-1} i_p \cdot (n+1)^p$. Let $h^{-1}$ denote the *inverse* mapping of $h$, i.e. $h(X) = i \Leftrightarrow h^{-1}(i) = X$. Observe that $X \le Y$ implies that $h(X) \le h(Y)$, though the opposite is not necessary true (i.e. it is possible that $i \le j$ and yet $h^{-1}(i) \nleq h^{-1}(j)$, as in the example in Figure 15b).

Each value of the form $\beta_{X,\ Y}$ will be computed and stored in a corresponding entry $B_{i,\ j}$, where $i = h(X)$ and $j = h(Y)$. Entries of $B$ which do not correspond to valid sub-instances of $S$, i.e. entries $B_{i,\ j}$ such that $h^{-1}(i) \nleq h^{-1}(j)$, will hold the value $\varphi$. The matrix $B$ is computed by applying the Inside VMT algorithm (Table 2) with a simple modification: in the base-cases of the recurrence (line 2 in Procedure Compute-Inside-Sub-Matrix, Table 2), the condition for computing the entry $B_{i,\ j}$ is that $h^{-1}(i) \le h^{-1}(j)$ rather than $i \le j$. If $h^{-1}(i) \le h^{-1}(j)$, the property $\beta_{h^{-1}(i),h^{-1}(j)}$ is computed and stored in this entry, and otherwise the entry retains its initial value $\varphi$.

In order to prove the correctness of the algorithm, we only need to show that all necessary values for computing a property $\beta_{X,\ Y}$ are available to the algorithm when the base-case of computing $B_{i,\ j}$ for $i = h(X)$ and $j = h(Y)$ is reached. Due to the nature of the mapping $h$, all properties $\beta_{X',Y'}$ for strict sub-instances $S_{X',Y'}$ of $S_{X,\ Y}$ correspond to entries $B_{i',\ j'}$ such that $i' = h(X')$ and $j' = h(Y')$ and $i', j' \in [i, j]$. Therefore, all inside properties of strict sub-instances of $S_{X,\ Y}$ are available according to the precondition. In addition, at this stage $B_{i,\ j}$ contains the value $B_{i,(i,j)} \times B_{(i,j),j}$. Note that for every $Q \in (X, Y)$, $q = h(Q) \in (i, j)$ (Figure 16a), and thus $\beta_{X,\ Q} \otimes \beta_{Q,\ Y} = B_{i,\ q} \otimes B_{q,\ j}$. On the other hand, every $q \in (i, j)$ such that $Q = h^{-1}(q) \notin (X, Y)$ sustains that either $X \nleq Q$ or $Q \nleq Y$ (Figure 16b), and therefore either $B_{i,\ q} = \varphi$ or $B_{q,\ j} = \varphi$, and $B_{i,\ q} \otimes B_{q,\ j} = \varphi$. Thus, $B_{i,\ (i,\ j)} \times B_{(i,j),j} = \oplus_{Q\in(X,Y)} (\beta_{X,\ Q} \otimes \beta_{Q,\ Y}) = \mu_{X,\ Y}$, and according to the Multiple String VMT requirements, the algorithm can compute $\beta_{X,\ Y}$ in $o(||S||)$ running time.

The following theorem summarizes the main result of this section.

**Theorem 3** *For every Multiple String (Inside or Outside) VMT problem there is an algorithm whose running time over an instance S is* $o(||S||^3)$. *When the problem sustains the standard VMT settings, the running time of the algorithm is* $\Theta(M(||S||))$, *where* $M(n)$ *is the running time of the corresponding matrix multiplication algorithm over two* $n \times n$ *matrices.*

## 6 Concluding remarks

This paper presents a simplification and a generalization of Valiant's technique, which speeds up a family of algorithms by incorporating fast matrix multiplication procedures. We suggest generic templates that identify problems for which the approach is applicable, where these templates are based on general recursive properties of the problems, rather than on their specific algorithms. Generic algorithms are described for solving all problems sustaining these templates.

The presented framework yields new worst case running time bounds for a family of important problems. The examples given here come from the fields of RNA secondary structure prediction and CFG Parsing. Recently, we have shown that this technique also applies for the *Edit Distance with Duplication and Contraction* problem [45], suggesting that it is possible that more problems from other domains can be similarly accelerated. While previous works describe other practical acceleration techniques for some of the mentioned problems [14,25,28,32-38], Valiant's technique, along with the Four Russians technique [28,30,31], are the only two techniques which currently allow to reduce the theoretical asymptotic worst case running times of the standard algorithms, without compromising the correctness of the computations.

The usage of the Four Russians technique can be viewed as a special case of using Valiant's technique. Essentially, the Four Russians technique enumerates solutions for small computations, stores these solutions

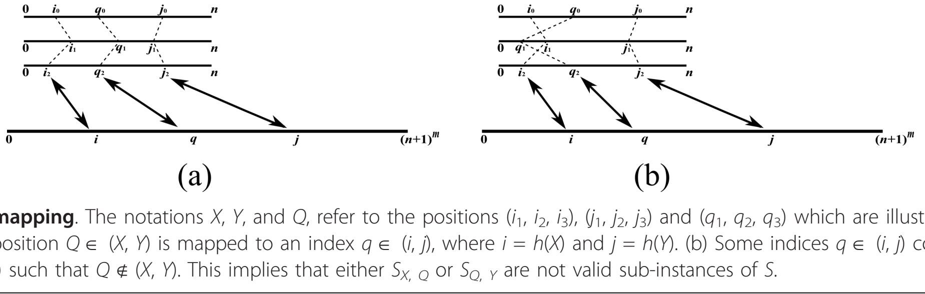


**Figure 16 The *h* mapping**. The notations $X$, $Y$, and $Q$, refer to the positions $(i_1, i_2, i_3)$, $(j_1, j_2, j_3)$ and $(q_1, q_2, q_3)$ which are illustrated in the figure, respectively. (a) A position $Q \in (X, Y)$ is mapped to an index $q \in (i, j)$, where $i = h(X)$ and $j = h(Y)$. (b) Some indices $q \in (i, j)$ correspond to positions $Q = h^{-1}(q)$ such that $Q \notin (X, Y)$. This implies that either $S_{X,\ Q}$ or $S_{Q,\ Y}$ are not valid sub-instances of $S$.

in lookup tables, and then accelerates bigger computations by replacing the execution of sub-computations with queries to the lookup tables. In the original paper [29], this technique was applied to obtain an $O(\log n)$ improvement factor for Boolean Matrix Multiplication. This approach was modified in [30] in order to accelerate Min/Max-Plus Multiplication, for integer matrices in which the difference between adjacent cells are taken from a finite interval of integers. While in [30] this technique was used ad hoc for accelerating RNA folding, it is possible to decouple the description of the fast matrix multiplication technique from the RNA folding algorithm, and present the algorithm of [30] in the same framework presented here. The latter special matrix multiplication variant was further accelerated in [45] to $O\left(\frac{n^3}{\log^2 n}\right)$ running time, implying that RNA folding under discrete scoring schemes (e.g. [6]) can be computed in $O\left(\frac{n^3}{\log^2 n}\right)$ time, instead of in $O\left(\frac{n^3 \log^3 \log n}{\log^2 n}\right)$ time as implied by the fastest Min/Max-Plus Multiplication algorithm for general matrices [27].

Many of the current acceleration techniques for RNA and CFG related algorithms are based on sparsification, and are applicable only to optimization problems. Another important advantage of Valiant's technique is that it is the only technique which is currently known to reduce the running times of algorithms for the non-optimization problem variants, such as RNA Partition Function related problems [10,12] and the WCFG Inside-Outside algorithm [43], in which the goals are to compute the sum of scores of all solutions of the input, instead of computing the score of an optimal solution.

Our presentation of the algorithm also improves upon previous descriptions in additional aspects. In Valiant's original description there was some intersection between the inputs of recursive calls in different branches of the recurrence tree, where portions of the data were re-computed more than once. Following Rytter's description of the algorithm [42], our formulation applies the recurrence on mutually-exclusive regions of the data, in a classic divide-and-conquer fashion. The formulation is relatively explicit, avoiding reductions to problems such as transitive closure of matrix multiplication [23,26] or shortest paths on lattice graphs [42]. The requirements specified here for VMT problems are less strict than requirements presented in previous works for such problems. Using the terminology of this work, additional requirements in [23] for Inside VMT problems are that the $\oplus$ operation of the multiplication is commutative, and that the $\oplus$ operation distributes over $\oplus$. Our explicit formulation of the algorithm makes it easy to observe that none of the operations of the presented algorithm requires the assumption that $\oplus$ distributes over $\oplus$. In addition, it can be seen that the $\oplus$ operation is always applied in a left-to-right order (for non-Multiple String VMT problems), thus the computation is correct even if $\oplus$ is not commutative. In [42], it was required that the algebraic structure $(\mathcal{D}, \oplus, \otimes)$ would be a *semiring*, i.e. adding to the requirements of [23] an additional requirement that $\mathcal{D}$ also contains an identity element 1 with respect to the $\oplus$ operation. Again, the algorithms presented here do not require that such a property be obeyed.

The time complexities of VMT algorithms are dictated by the time complexities of matrix multiplication algorithms. As matrix multiplication variants are essential operations in many computational problems, much work has been done to improve both the theoretical and the practical running times of these operations, including many recent achievements [24,27,40,41,46-50]. Due to its importance, it is expected that even further improvements in this domain will be developed in the future, allowing faster implementations of VMT algorithms. Theoretical sequential sub-cubic matrix multiplication algorithms (e.g. [24]) are usually considered impractical for realistic matrix sizes. However, practical, hardware-based fast computations of matrix multiplications are gaining popularity within recent years [40,41], due the highly parallelized nature of such computations and the availability of new technologies that exploit this parallelism. Such technologies were previously used for some related problems [51,52], yet there is an intrinsic advantage for its utilization via the VMT framework. While optimizing the code for each specific problem and each specific hardware requires special expertise, the VMT framework conveniently allows differing the bottleneck part of the computation to the execution of matrix multiplication subroutines, and thus off-the-shelf, hardware tailored optimized solutions can be easily integrated into all VMT problems, instead of being developed separately for each problem.

## Appendix

### A RNA Partition Function and Base Pair Binding Probabilities

The following example is a simplified formalization of the *RNA Partition Function and Base Pair Binding Probabilities* problem [10]. The example demonstrates the Inside-Outside VMT settings, with respect to the Dot Product variant. This problem requires the computation of several inside and outside properties for every sub-instance of the input.

As in Section 3.1, we consider the RNA folding domain. Here, instead of looking for an *optimal* folding of an input RNA string $s$, we attempt to answer the following questions:

1. For a given folding $F$ of $s$, what is the probability that $s$ folds spontaneously to $F$?
2. For every pair of indices $a$ and $b$, what is the probability that a spontaneous folding of $s$ contains the base-pair $a \bullet b$?

The *Partition Function* physical model allows answering these questions. When applied to RNA folding, it is assumed that for every folding $F$ of $s$, it is possible to assign a *weight* $w(F) = e^{\frac{-E(F)}{k_B T}}$, which is proportional to the *probability* that $s$ spontaneously folds according to $F$. Here, $E(F)$ is the energy of $F$, $k_B$ is the Boltzmann constant, and $T$ is the temperature. Then, the probability that $s$ folds according to $F$ is given by $\frac{w(F)}{Z}$, where the normalizing factor $Z$ is called the *Partition Function* and is given by

$$Z = \sum_{\substack{F' \text{ a folding} \\ \text{of } s}} w(F').$$

Note that a naive computation of the RNA folding partition function would require the examination of all possible foldings of the input instance, where the number of such foldings grows exponentially with the length of the instance [53]. In [10], McCaskill showed how to efficiently compute the partition function for RNA folding in $\Theta(n^3)$ running time. In addition, he presented a variant that allows the computation of base-pairing probabilities for every given pair of indices in the RNA string. We next present a simplified version of McCaskill's computation. For the sake of clarity of presentation, assume that $w(F) = e^{|F|}$, where $|F|$ is the number of base-pairs in $F$, and assume that non-complementary base-pairs are not allowed to occur. This simplification of the weight function allows focusing on the essential properties of the computation, avoiding a tedious notation.

For the "inside" phase of the computation, define two sets of inside properties $\beta^1$ and $\beta^2$ with respect to an input RNA string $s$. The property $\beta^2_{i,j}$ is the partition function of the substring $s_{i,j}$, i.e. the sum of weights of all possible foldings of $s_{i,j}$. The property $\beta^1_{i,j}$ is the sum of weights of all possible foldings of $s_{i,j}$ that contain the base-pair $i \bullet (j - 1)$. For $j \leq i + 1$, the only possible folding of $s_{i,j}$ is the empty folding, and thus $\beta^1_{i,j} = 0$ (since there are no foldings of $s_{i,j}$ that contain the base-pair $i \bullet (j - 1)$) and $\beta^2_{i,j} = 1$ (since the weight of the empty folding is $e^0 = 1$). For $j > i + 1$, $\beta^1_{i,j}$ and $\beta^2_{i,j}$ can be recursively computed as follows.

First, consider the computation of $\beta^1_{i,j}$. Observe that if $s_i$ and $s_{j-1}$ are not complementary bases, then there is no folding of $s_{i,j}$ that contains $i \bullet (j - 1)$, and thus $\beta^1_{i,j} = 0$. Otherwise, every folding $F$ of $s_{i,j}$ that contains $i \bullet (j - 1)$ is obtained by adding the base-pair $i \bullet (j - 1)$ to a unique folding $F'$ of $s_{i+1,j-1}$, where $w(F) = e^{|F|} = e^{|F'|+1} = e \cdot e^{|F'|}$. Therefore, $\beta^1_{i,j}$ is given by

$$\beta^1_{i,j} = \delta_{i,j-1}\beta^2_{i+1,j-1},$$

where $\delta_{i,j-1} = e$ if $s_i$ and $s_{j-1}$ are complementary bases, and otherwise $\delta_{i,j-1} = 0$.

Now, consider the computation of $\beta^2_{i,j}$. Every folding $F$ of $s_{i,j}$ in which $j - 1$ is paired to some index $q$, $i < q < j - 1$, is obtained by combining a folding $F'$ of the prefix $s_{i,q}$ with a folding $F''$ of the suffix $s_{q,j}$, where $F''$ contains the base-pair $q \bullet (j - 1)$. In addition $w(F) = e^{|F|} = e^{|F'|+|F''|} = e^{|F'|} \cdot e^{|F''|} = w(F') \cdot w(F'')$. Thus, the sum of weights of all possible different foldings of this form, denoted by $\mu^2_{i,j}$, is given by

$$\mu^2_{i,j} = \sum_{q \in (i,j-1)} \left\{ \sum_{\substack{F' \text{ a folding} \\ \text{of } s_{i,q}}} \left\{ \sum_{\substack{F'' \text{ a folding of} \\ s_{q,j} \text{ which} \\ \text{contains } q \bullet (j-1)}} \{w(F') \cdot w(F'')\} \right\} \right\} = \sum_{q \in (i,j-1)} \{\beta^2_{i,q} \cdot \beta^1_{q,j}\}.$$

Since $\beta^1_{j-1,j} = 0$, $\mu^2_{i,j}$ can be written as $\mu^2_{i,j} = \sum_{q \in (i,j)} \{\beta^2_{i,q} \cdot \beta^1_{q,j}\}$.

In addition to foldings of the above form, the set of foldings of $s_{i,j}$ contains foldings in which $j - 1$ is unpaired, and foldings in which $j - 1$ is paired to $i$. The set of foldings of $s_{i,j}$ in which $j - 1$ is unpaired is exactly the set of foldings of $s_{i,j-1}$, and their sum of weights is given by $\beta^2_{i,j-1}$. The sum of weights of all foldings in which $j - 1$ is paired to $i$ is given by $\beta^1_{i,j}$, and thus

$$\beta^2_{i,j} = \mu^2_{i,j} + \beta^2_{i,j-1} + \beta^1_{i,j}.$$

The computation of the inside property sets $\beta^1$ and $\beta^2$ abides by the Inside VMT requirements of Definition 1 with respect to the Dot Product, and thus may be computed by the Inside VMT algorithm. The partition function of the input RNA string $s$ is given by $Z = \beta^2_{0,n}$.

The second phase of McCaskill's algorithm computes for every pair of indices $a$ and $b$ in $s$, the probability that a spontaneous folding of $s$ contains the base-pair $a \bullet b$. According to the partition function model, this probability equals to the sum of weights of foldings of $s$ which contain $a \bullet b$, divided by $Z$. Therefore, there is a need to compute values that correspond to sums of weights of foldings which contain specific base-pairs. We will denote by $\gamma_{i,\,j}$ the sum of weights of foldings of $s$ which contain the base pair $i \bullet (j - 1)$. The probability of a base-pair $a \bullet b$ is thus $\frac{\gamma_{a,b+1}}{Z}$. In order to compute values of the form $\gamma_{i,\,j}$, we define the following *outside* property sets $\alpha^1$, $\alpha^2$, $\alpha^3$ and $\alpha^4$.

A value of the form $\alpha^1_{i,j}$ reflects the sum of weights of all foldings of $\overline{s_{i,j}}$ that contain the base-pair $(i - 1) \bullet j$. A value of the form $\alpha^2_{i,j}$ reflects the sum of weights of all foldings of $\overline{s_{i,j}}$ that contain a base-pair of the form $(q - 1) \bullet j$, for some index $q \in [0, i)$. A value of the form $\alpha^3_{i,j}$ reflects the sum of weights of all foldings of $\overline{s_{i,j}}$ that contain a base-pair of the form $j \bullet (q - 1)$, for some index $q \in (j, n]$, and a value of the form $\alpha^4_{i,j}$ reflects the partition function of $\overline{s_{i,j}}$, i.e. the sum of weights of all foldings of $\overline{s_{i,j}}$.

Every folding $F$ for $s$ in which the base-pair $i \bullet (j - 1)$ is included can be decomposed into two foldings $F'$ and $F''$, where $F'$ is a folding of $s_{i,\,j}$ that contains $i \bullet (j - 1)$, and $F''$ is a folding of $\overline{s_{i,j}}$. Similarly as above, this observation implies that the sum of weights of all foldings of $s$ that contain $i \bullet (j - 1)$ is

$$\gamma_{i,j} = \beta^1_{i,j} \cdot \alpha^4_{i,j}.$$

It is now left to show how to compute values of the form $\alpha^4_{i,j}$. This computation is showed next, via a mutually recursive formulation that also computes values in the sets $\alpha^1$, $\alpha^2$, and $\alpha^3$.

Every folding of $\overline{s_{i,j}}$ that contains $(i - 1) \bullet j$ is obtained by adding the base-pair $(i - 1) \bullet j$ to a unique folding of $\overline{s_{i-1,j+1}}$. As before, this implies that

$$\alpha^1_{i,j} = \delta_{i-1,j} \cdot \alpha^4_{i-1,j+1}.$$

Every folding of $\overline{s_{i,j}}$ that contains a base-pair of the form $(q - 1) \bullet j$, for some $q \in [0, i)$, can be decomposed into two foldings $F'$ and $F''$, where $F'$ is a folding of $s_{q,\,i}$, and $F''$ is a folding of $\overline{s_{q,j}}$ that contains $(q - 1) \bullet j$. Again, this implies that the sum of weights of all such foldings is given by

$$\alpha^2_{i,j} = \sum_{q \in [0,i)} \{\beta^2_{q,i} \cdot \alpha^1_{q,j}\}.$$

Similarly, it can be shown that

$$\alpha^3_{i,j} = \sum_{q \in (j,n]} \{\alpha^4_{i,q} \cdot \beta^1_{j,q}\}.$$

Finally, consider the computation of $\alpha^4_{i,j}$. Note that the set of all foldings of $\overline{s_{i,j}}$ in which $j$ is unpaired is exactly the set of all foldings of $\overline{s_{i,j+1}}$, and thus the sum of weights of all such foldings is $\alpha^4_{i,j+1}$. In every other folding of $\overline{s_{i,j}}$, $j$ is either paired to $i - 1$, or to some index $q - 1$ such that $q \in [0, i)$ or $q \in (j, n]$. Therefore,

$$\alpha^4_{i,j} = \alpha^4_{i,j+1} + \alpha^1_{i,j} + \alpha^2_{i,j} + \alpha^3_{i,j}.$$

It can now be verified that the computation of the property sets $\alpha^1$, $\alpha^2$, $\alpha^3$, and $\alpha^4$ sustains all requirements from an Outside VMT problem as listed in Definition 2, therefore the Base Pair Binding Probabilities problem may be computed by the Outside-VMT algorithm.

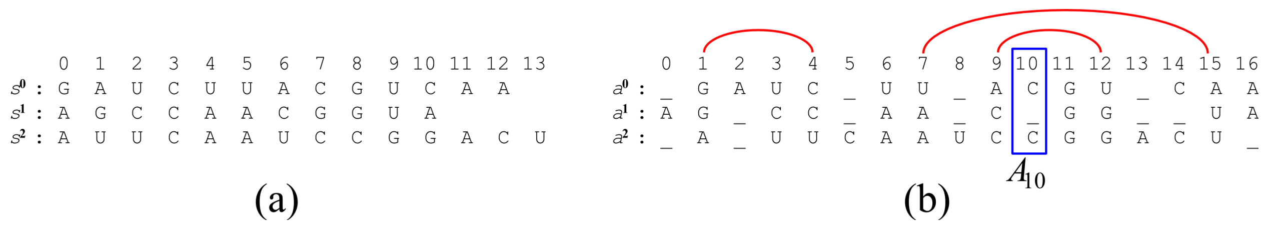
## B RNA Simultaneous Alignment and Folding

The *RNA Simultaneous Alignment and Folding (SAF)* problem is an example of a Multiple String VMT problem, defined by Sankoff [15]. Similarly to the classical sequence alignment problem, the goal of the SAF problem is to find an alignment of several RNA strings, and in addition to find a common folding for the aligned segments of the strings. The score of a given alignment with folding takes into account both standard alignment elements such as character matchings, substitutions and indels, as well as the folding score. For clarity, our formulation assumes a simplified scoring scheme.

We use the notation of Section 5 regarding multiple string instances, positions, and sub-instances. Let $Q = (q^0, q^1, ..., q^{m-1})$ be a position in a multiple string instance $S = (s^0, s^1, ..., s^{m-1})$. Say that a position $Q' = (q'^0, q'^1, ..., q'^{m-1})$ in $S$ is a *local increment* of $Q$ if $Q < Q'$, and for every $0 \le p < m$, $q'^p \le q^p + 1$. That is, a local increment of a position increases the value of each one of the sequence indices of the position by at most 1, where at least one of the indices strictly increases. Symmetrically, say that in the above case $Q$ is a *local decrement* of $Q'$. The position sets $inc(Q)$ and $dec(Q)$ denote the set of all local increments and the set of all local decrements of $Q$, respectively. The size of each one of these sets is $O(2^m)$. An *SAF instance* $S$ is a set of RNA strings $S = (s^0, s^1, ..., s^{m-1})$. An *alignment* $A$ of $S$ is a set of strings $A = (a^0, a^1, ..., a^{m-1})$ over the alphabet $\{A, C, G, U, -\}$ ('-' is called the *"gap"* character), satisfying that:

- All strings in $A$ are of the same length.
- For every $0 \le p < m$, the string which is obtained by removing from $a^p$ all gap characters is exactly the string $s^p$.

Denote by $|A|$ the length of each one of the strings in $A$, and by $A_r = (a^0_r; a^1_r; \ldots; a^{m-1}_r)$ the $r$th *column* of $A$. Observe that an alignment column $A_r$ corresponds to a sub-instance of the form $S_{Q,\,Q'}$, where $Q'$ is a local



0 1 2 3 4 5 6 7 8 9 10 11 12 13
s⁰ : G A U C U U A C G U C A A
s¹ : A G C C A A C G G U A
s² : A U U C A A U C C G G A C U
(a)
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
a⁰ : _ G A U C _ U U _ A C G U _ C A A
a¹ : A G _ C C _ A A _ C _ G G _ _ U A
a² : _ A _ U U C A A U C C G G A C U _
A10
(b)

**Figure 17 An SAF instance, and a corresponding alignment with folding**. (a) An SAF instance *S*, composed of three RNA strings $s^0$, $s^1$, and $s^2$. (b) A possible alignment with folding (*A*, *F*) of *S*, where $|A| = 17$, and $F = \{1 \bullet 4, 7 \bullet 15, 9 \bullet 12\}$. The alignment column $A_{10}$ is marked. This column corresponds to the sub-instance $S_{Q,\ Q'}$, for $Q = (7, 7, 8)$ and $Q' = (8, 7, 9)$.

increment of $Q$. The position $Q = (q^0, q^1, ..., q^{m-1})$ in $S$ satisfies that for every $0 \le p < m$, $q^p$ is the number of non-gap characters in $a^p_{0,r}$. The position $Q' = (q'^0, q'^1, ..., q'^{m-1})$ satisfies that for every $0 \le p < m$, $q'^p = q^p$ if $a^p_r$ is a gap, and otherwise $q'^p = q^p + 1$ (see Figure 17).

A *folding F* of an alignment *A* is defined similarly to a folding of a single RNA string, except for the fact that now each pair $a \bullet b$ in $F$ represents a column-pair in an alignment rather than a base-pair in an RNA string. Call a pair $(A, F)$, where $A$ is an alignment of an SAF instance $S$ and $F$ is a folding of $A$, an *alignment with folding* of $S$ (Figure 17b). Define the score of an alignment with folding $(A, F)$ to be

$$score\ (A,\ F) = \sum_{0 \le r < |A|} \rho(A_r) + \sum_{a \bullet b \in F} \tau(A_a,\ A_b)$$

Here, $\rho$ is a *column aligning score function*, and $\tau$ is a *column-pair aligning score function*. $\rho\ (A_r)$ reflects the alignment quality of the $r$th column in $A$, giving high scores for aligning nucleotides of the same type and penalizing alignment of nucleotides of different types or aligning nucleotides against gaps. $\tau(A_a, A_b)$ reflects the benefit from forming a base-pair in each one of the RNA strings in $S$ between the bases corresponding to columns $A_a$ and $A_b$ of the alignment (if gaps or non-complementary bases are present in these columns, it may induce a score penalty). In addition, compensatory mutations in these columns may also increase the value of $\tau(A_a, A_b)$ (thus it may compensate for some penalties taken into account in the computation of $\rho(A_a)$ and $\rho(A_b)$). We assume that both scoring functions $\rho$ and $\tau$ can be computed in $O(m)$ running time. In addition, we use as arguments for $\rho$ and $\tau$ sub-instances of the form $S_{Q,\ Q'}$, where $Q'$ is a local increment of $Q$. In such cases, $S_{Q,\ Q'}$ corresponds to a unique alignment column, where $\rho$ and $\tau$ are computed with respect to this column.

The goal of the SAF problem is to find the maximum score of an alignment with folding for a given SAF instance $S$. In order to compute this value, we define a set $\beta$ of inside properties with respect to $S$, where $\beta_{X,\ Y}$ is the maximum score of an alignment with folding of $S_{X,\ Y}$.

Similarly to single RNA string folding (Section 3.1), we think of two kinds of alignments with foldings for the sub-instance $S_{X,\ Y}$ : type *I* are alignments with foldings in which the first and last alignment columns are paired to each other, and type *II* are alignments with foldings in which the first and last alignment columns are not paired to each other.

Consider an alignment with folding $(A, F)$ of type *I*. Let $A_a$ denote the first column in $A$, and $A_b$ the the last column in $A$. Thus, $A_a$ corresponds to some sub-instance $S_{X,\ X'}$, where $X'$ is a local increment of $X$, and similarly $A_b$ corresponds to some sub-instance $S_{Y',Y}$, where $Y'$ is a local decrement of $Y$. Since $a \bullet b \in F$, the alignment with folding $(A, F)$ is obtained by modifying an alignment with folding $(A', F')$ for $S_{X',Y'}$, where $A$ is obtained by concatenating $A_a$ and $A_b$ at the beginning and ending of $A'$, respectively, and $F = \{a \bullet b\} \cup F'$ (Figure 18). The score of $(A, F)$ is given by

$$score\ (A,\ F) = score(A',\ F') + \rho(S_{X,X'}) + \rho(S_{Y',Y}) + \tau(S_{X,X'},\ S_{Y',Y}).$$

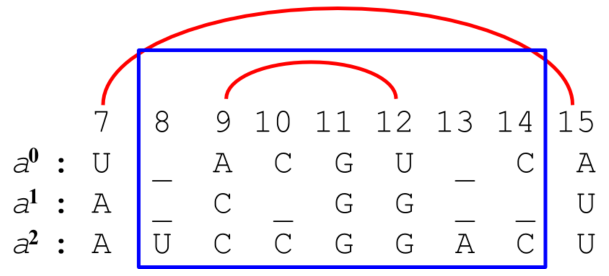


**Figure 18 An alignment with folding of type *I***. The alignment with folding (*A*, *F*) corresponds to the sub-instance $S_{X,\ Y}$ of the instance *S* presented in Figure 17a, where $X = (5, 5, 5)$ and $Y = (12, 10, 14)$. (*A*, *F*) is obtained by modifying an alignment with folding (*A'*, *F'*) for the sub-instance $S_{X',Y'}$, where $X' = (6, 6, 6)$ and $Y' = (11, 9, 13)$ (marked with a blue rectangle). The modification includes the addition of the two alignment columns $A_7$ and $A_{15}$ to the alignment, and the column-pair $7 \bullet 15$ to the folding.

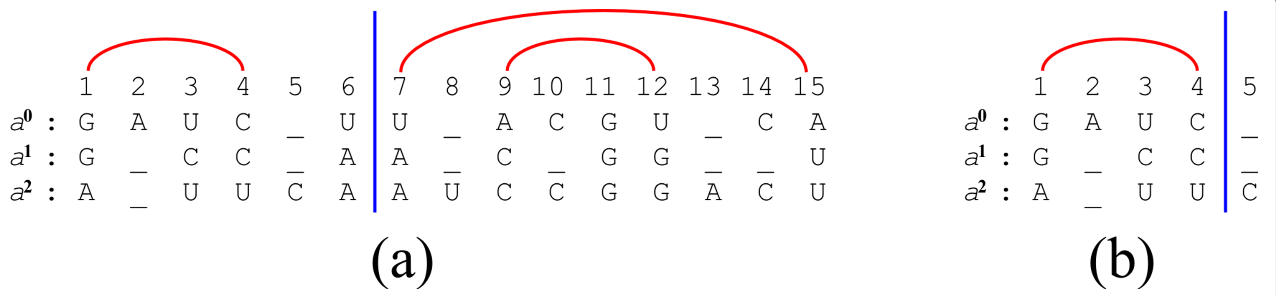

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
a⁰ : G A U C _ U U _ A C G U _ C A
a¹ : G _ C C _ A A _ C _ G G _ _ U
a² : A _ U U C A A U C C G G A C U
(a)
1 2 3 4 5
a⁰ : G A U C _
a¹ : G _ C C _
a² : A _ U U C
(b)

**Figure 19 An alignment with folding of type *II*.** In such an alignment with folding the first and last alignment columns are not paired to each other, i.e. the last column is either paired to some other column (as in (a), which consists of columns 1 to 15 of the alignment with folding presented in Figure 17b), or it is unpaired (as in (b), which consists of columns 1 to 5 of the alignment with folding presented in Figure 17b). In both cases, the alignment with folding may be decomposed into two alignments with foldings (*A'*, *F'*) and (*A''*, *F''*), where (*A'*, *F'*) is an alignment with folding for $S_{X,Q}$ and (*A''*, *F''*) is an alignment with folding for $S_{Q,Y}$, for some $Q \in (X, Y)$.

Therefore, it can be seen that the maximum score of an alignment with folding of type *I* is given by

$$\max_{\substack{X' \in inc(X)\\ Y' \in dec(Y)}} \{\beta_{X',Y'} + \rho(S_{X,X'}) + \rho(S_{Y',Y}) + \tau(S_{X,X'}, S_{Y',Y})\}.$$

Now, consider an alignment with folding $(A, F)$ of type *II*. As was shown for the Base-Paring Maximization problem (Section 3.1), $(A, F)$ can be decomposed into two alignments with folding $(A', F')$ and $(A'', F'')$, where $(A', F')$ is an alignment with folding of $S_{X,Q1}$ and $(A'', F'')$ is an alignment with folding of $S_{Q,Y}$, for some $Q \in (X, Y)$ (Figure 19). In this case, $score(A, F) = score(A', F') + score(A'', F'')$, and the maximum score of an alignment with folding of type *II* is

$$\max_{Q \in (X,Y)} \{\beta_{X,Q} + \beta_{Q,Y}\}.$$

Thus, $\beta_{X,Y}$ can be recursively computed according to the following formula:

$$\beta_{X,Y} = \max \left\{ \begin{array}{lll} (I) & \max\limits_{\substack{X' \in inc(X)\\ Y' \in dec(Y)}} & \{\beta_{X',Y'} + \rho(S_{X,X'}) + \rho(S_{Y',Y}) + \tau(S_{X,X'}, S_{Y',Y})\}, \\ (II) & \max\limits_{Q \in (X,Y)} & \{\beta_{X,Q} + \beta_{Q,Y}\} \end{array} \right\}.$$

In the computation of term $(I)$ of the recurrence, $O(2^{2m})$ expressions are examined, and each expression is computed in $O(m)$ running time. Under the reasonable assumption that $m$ is sufficiently small with respect to $||S||$ (typically, $m = 2$), we can assume that $m2^{2m} = o(||S||)$, and even assume that $m2^{2m} = O\left(\frac{M(||S||)}{||S||^2}\right)$, where $M(r)$ is the running time of the Max Plus multiplication of two $r \times r$ matrices.

The computation of term $(II)$ is a Max Plus vector multiplication of the form $\mu_{X,Y} = \oplus_{Q \in (X,Y)} (\beta_{X,Q} \otimes \beta_{Q,Y})$, and thus the recursive formulation abides by the standard VMT settings for the Multiple String Inside VMT requirements, as listed in Definition 3. Therefore, the SAF problem with an instance $S$ can be solved in $\Theta(M(||S||)) = o(||S||^3)$ running time. This result improves the running time of the original algorithm [15], which is $\Theta(||S||^3)$.

**Acknowledgements**
The research of SZ and MZU was supported by ISF grant 478/10.

**Authors' contributions**
SZ developed and formulated the algorithms, template definitions, and examples. MZU and DT mentored the research, contributed in writing parts of the manuscript, and were active in revising and preparing the final manuscript. All authors read and approved the final manuscript.

**Competing interests**
The authors declare that they have no competing interests.

Received: 10 November 2010 Accepted: 18 August 2011
Published: 18 August 2011

## References

1. Eddy SR: **Noncoding RNA genes.** *Current Opinions in Genetic Development* 1999, **9(6)**:695-699[http://view.ncbi.nlm.nih.gov/pubmed/10607607].
2. Mandal M, Breaker R: **Gene regulation by riboswitches.** *Cell* 2004, **6**:451-463.
3. Griffiths-Jones S, Moxon S, Marshall M, Khanna A, Eddy S, Bateman A: **Rfam: annotating non-coding RNAs in complete genomes.** *Nucleic Acids Research* 2005, , **33 Database:** D121.
4. Consortium AFB, Backofen R, Bernhart SH, Flamm C, Fried C, Fritzsch G, Hackermuller J, Hertel J, Hofacker IL, Missal K, Mosig A, Prohaska SJ, Rose D, Stadler PF, Tanzer A, Washietl S, Will S: **RNAs everywhere: genome-wide annotation of structured RNAs.** *J Exp Zoolog B Mol Dev Evol* 2007, **308**:1-25.
5. Gardner P, Giegerich R: **A comprehensive comparison of comparative RNA structure prediction approaches.** *BMC bioinformatics* 2004, **5**:140.
6. Nussinov R, Jacobson AB: **Fast Algorithm for Predicting the Secondary Structure of Single-Stranded RNA.** *PNAS* 1980, **77(11)**:6309-6313.
7. Zuker M, Stiegler P: **Optimal Computer Folding of Large RNA Sequences using Thermodynamics and Auxiliary Information.** *Nucleic Acids Research* 1981, **9**:133-148.
8. Hofacker IL, Fontana W, Stadler PF, Bonhoeffer SL, Tacker M, Schuster P: **Fast Folding and Comparison of RNA Secondary Structures.** *Monatsh Chem* 1994, **125**:167-188.
9. Alkan C, Karakoç E, Nadeau JH, Sahinalp SC, Zhang K: **RNA-RNA Interaction Prediction and Antisense RNA Target Search.** *Journal of Computational Biology* 2006, **13(2)**:267-282.
10. McCaskill JS: **The equilibrium partition function and base pair binding probabilities for RNA secondary structure.** *Biopolymers* 1990, **29(6-7)**:1105-1119.

11. Bernhart S, Tafer H, Mückstein U, Flamm C, Stadler P, Hofacker I: **Partition function and base pairing probabilities of RNA heterodimers.** *Algorithms for Molecular Biology* 2006, **1**:3.
12. Chitsaz H, Salari R, Sahinalp SC, Backofen R: **A partition function algorithm for interacting nucleic acid strands.** *Bioinformatics* 2009, **25(12)**:i365-373.
13. Zhang K: **Computing Similarity Between RNA Secondary Structures.** *INTSYS '98: Proceedings of the IEEE International Joint Symposia on Intelligence and Systems* Washington, DC, USA: IEEE Computer Society; 1998, 126.
14. Jansson J, Ng S, Sung W, Willy H: **A faster and more space-efficient algorithm for inferring arc-annotations of RNA sequences through alignment.** *Algorithmica* 2006, **46(2)**:223-245.
15. Sankoff D: **Simultaneous Solution of the RNA Folding, Alignment and Protosequence Problems.** *SIAM Journal on Applied Mathematics* 1985, **45(5)**:810-825.
16. Sakakibara Y, Brown M, Hughey R, Mian I, Sjolander K, Underwood R, Haussler D: **Stochastic context-free grammers for tRNA modeling.** *Nucleic Acids Research* 1994, **22(23)**:5112.
17. Teitelbaum R: **Context-Free Error Analysis by Evaluation of Algebraic Power Series.** *STOC ACM* 1973, 196-199.
18. Dowell R, Eddy S: **Evaluation of several lightweight stochastic context-free grammars for RNA secondary structure prediction.** *BMC bioinformatics* 2004, **5**:71.
19. Do CB, Woods DA, Batzoglou S: **CONTRAfold: RNA secondary structure prediction without physics-based models.** *Bioinformatics* 2006, **22(14)**:e90-8.
20. Cocke J, Schwartz JT: *Programming Languages and Their Compilers* New York: Courant Institute of Mathematical Sciences; 1970.
21. Kasami T: **An efficient recognition and syntax analysis algorithm for context-free languages.** *Tech. Rep. AFCRL-65-758, Air Force Cambridge Res. Lab., Bedford Mass* 1965.
22. Younger DH: **Recognition and Parsing of Context-Free Languages in Time $n^3$.** *Information and Control* 1967, **10(2)**:189-208.
23. Valiant L: **General Context-Free Recognition in Less than Cubic Time.** *Journal of Computer and System Sciences* 1975, **10**:308-315.
24. Coppersmith D, Winograd S: **Matrix Multiplication via Arithmetic Progressions.** *J Symb Comput* 1990, **9(3)**:251-280.
25. Akutsu T: **Approximation and Exact Algorithms for RNA Secondary Structure Prediction and Recognition of Stochastic Context-free Languages.** *Journal of Combinatorial Optimization* 1999, **3**:321-336.
26. Benedí J, Sánchez J: **Fast Stochastic Context-Free Parsing: A Stochastic Version of the Valiant Algorithm.** *Lecture Notes in Computer Science* 2007, **4477**:80-88.
27. Chan TM: **More Algorithms for All-Pairs Shortest Paths in Weighted Graphs.** *SIAM J Comput* 2010, **39(5)**:2075-2089.
28. Graham SL, Harrison MA, Ruzzo WL: **An improved context-free recognizer.** *ACM Transactions on Programming Languages and Systems* 1980, **2(3)**:415-462.
29. Arlazarov VL, Dinic EA, Kronod MA, Faradzev IA: **On Economical Construction of the Transitive Closure of an Oriented Graph.** *Soviet Math Dokl* 1970, **11**:1209-1210.
30. Frid Y, Gusfield D: **A Simple, Practical and Complete $O(\frac{n^3}{\log n})$-Time Algorithm for RNA Folding Using the Four-Russians Speedup.** In *WABI. Volume 5724.* Springer; 2009:97-107.
31. Frid Y, Gusfield D: **A Worst-Case and Practical Speedup for the RNA Co-folding Problem Using the *Four-Russians* Idea.** *WABI* 2010, 1-12.
32. Klein D, Manning CD: **A* Parsing: Fast Exact Viterbi Parse Selection.** *HLT-NAACL* 2003, 119-126.
33. Wexler Y, Zilberstein CBZ, Ziv-Ukelson M: **A Study of Accessible Motifs and RNA Folding Complexity.** *Journal of Computational Biology* 2007, **14(6)**:856-872.
34. Ziv-Ukelson M, Gat-Viks I, Wexler Y, Shamir R: **A Faster Algorithm for Simultaneous Alignment and Folding of RNA.** *Journal of Computational Biology* 2010, **17(8)**:1051-1065[http://www.liebertonline.com/doi/abs/10.1089/cmb.2009.0197].
35. Backofen R, Tsur D, Zakov S, Ziv-Ukelson M: **Sparse RNA folding: Time and space efficient algorithms.** *Journal of Discrete Algorithms* 2010, http://www.sciencedirect.com/science/article/B758J-511TNF7-1/2/8d480ed24b345199f8997c1141a47d60.
36. Salari R, Mohl M, Will S, Sahinalp S, Backofen R: **Time and Space Efficient RNA-RNA Interaction Prediction via Sparse Folding.** *RECOMB* 2010, **6044**:473-490.
37. Havgaard J, Lyngso R, Stormo G, Gorodkin J: **Pairwise local structural alignment of RNA sequences with sequence similarity less than 40%.** *Bioinformatics* 2005, **21(9)**:1815-1824.
38. Will S, Reiche K, Hofacker IL, Stadler PF, Backofen R: **Inferring Non-Coding RNA Families and Classes by Means of Genome-Scale Structure-Based Clustering.** *PLOS Computational Biology* 2007, **3(4)**:e65.
39. Zakov S, Tsur D, Ziv-Ukelson M: **Reducing the Worst Case Running Times of a Family of RNA and CFG Problems, Using Valiant's Approach.** *WABI* 2010, 65-77.
40. Ryoo S, Rodrigues CI, Baghsorkhi SS, Stone SS, Kirk DB, Hwu WmW: **Optimization principles and application performance evaluation of a multithreaded GPU using CUDA.** *Proceedings of the 13th ACM SIGPLAN Symposium on Principles and practice of parallel programming, PPoPP '08, New York, NY, USA: ACM* 2008, 73-82.
41. Volkov V, Demmel JW: **Benchmarking GPUs to tune dense linear algebra.** *Proceedings of the 2008 ACM/IEEE conference on Supercomputing* SC '08, Piscataway, NJ, USA: IEEE Press; 2008, 31:1-31:11[http://portal.acm.org/citation.cfm?id=1413370.1413402].
42. Rytter W: **Context-free recognition via shortest paths computation: a version of Valiant's algorithm.** *Theoretical Computer Science* 1995, **143(2)**:343-352.
43. Baker JK: **Trainable grammars for speech recognition.** *The Journal of the Acoustical Society of America* 1979, **65(S1)**:S132-S132.
44. Bentley JL, Haken D, Saxe JB: **A General Method For Solving Divide-and-conquer Recurrences.** *SIGACT News* 1980, **12(3)**:36-44.
45. Pinhas T, Tsur D, Zakov S, Ziv-Ukelson M: **Edit Distance with Duplications and Contractions Revisited.** In *CPM of Lecture Notes in Computer Science. Volume 6661.* Edited by: Giancarlo R, Manzini G. Springer Berlin/Heidelberg; 2011:441-454.
46. Goto K, Geijn R: **Anatomy of high-performance matrix multiplication.** *ACM Transactions on Mathematical Software (TOMS)* 2008, **34(3)**:1-25.
47. Robinson S: **Toward an optimal algorithm for matrix multiplication.** *News Journal of the Society for Industrial and Applied Mathematics* 2005, **38(9)**.
48. Basch J, Khanna S, Motwani R: **On diameter verification and boolean matrix multiplication.** *Tech rep Citeseer* 1995.
49. Williams R: **Matrix-vector multiplication in sub-quadratic time (some preprocessing required).** *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms, Society for Industrial and Applied Mathematics* 2007, 995-1001.
50. Bansal N, Williams R: **Regularity Lemmas and Combinatorial Algorithms.** *FOCS* 2009, 745-754.
51. Rizk G, Lavenier D: **GPU Accelerated RNA Folding Algorithm.** In *Computational Science - ICCS 2009, Volume 5544 of Lecture Notes in Computer Science.* Edited by: Allen G, Nabrzyski J, Seidel E, van Albada G, Dongarra J, Sloot P. Springer Berlin/Heidelberg; 1004-1013.
52. Chang D, Kimmer C, Ouyang M: **Accelerating the Nussinov RNA folding algorithm with CUDA/GPU.** *Signal Processing and Information Technology (ISSPIT), 2010 IEEE International Symposium on* IEEE;120-125.
53. Waterman M: **Secondary structure of single-stranded nucleic acids.** *Adv math suppl studies* 1978, **1**:167-212.

doi:10.1186/1748-7188-6-20
**Cite this article as:** Zakov *et al.*: Reducing the worst case running times of a family of RNA and CFG problems, using Valiant's approach. *Algorithms for Molecular Biology* 2011 **6**:20.