Speedup of RNA Pseudoknotted Secondary Structure Recurrence Computation with the Four-Russians Method

Yelena Frid and Dan Gusfield

Department of Computer Science, U.C. Davis

Abstract. While secondary pseudoknotted structure prediction is computationally challenging, such structures appear to play biologically important roles in both cells and viral RNA [1]. Restricting the class of possible structures and then finding the optimal structure for that restricted class is a common method employed to deal with the computational complexity.

We derive a practical and worst-case speedup algorithm using the Four-Russians method for the $O(n^6)$ time *Rivas&Eddy Algorithm* [2] describing the broadest set of structures. *Fast R&E* algorithm finds the optimal Rivas&Eddy fold in $O(n^6/q)$-time, where $q \geq log(n)$.

Because the solution matrix produced by *Fast R&E* algorithm is identical to the one produced by the original Rivas&Eddy algorithm, the contribution of the algorithm lies not only in its stand alone practicality but also in its ability to be implemented alongside heuristic speedups, leading to even greater reductions in time. Our approach is the first to achieve a $\Omega(log(n))$ time speedup without reducing the set of possible Rivas&Eddy pseudoknotted structures. The analysis presented here of the original algorithm could be used to improve other pseudoknot algorithms with similar recurrences.

1 Introduction

The algorithmic goal of structure prediction is motivated by the understanding that RNA structure helps to determine function. It has been particularly observed that in eukaryotic genomes ncRNA (Non-coding RNA) function is seen more clearly from structure [3–5]. Pseudoknot structures, specifically, play an important role in transcription regulation, as well as RNA splicing and catalysis [1, 6]. While algorithms that compute the optimal pseudoknot free fold for RNA[1][7] are solved in polynomial time $O(n^3)$ [7–9] or $O(n^3/log(n))$ [10] depending on problem formulation, the optimal secondary structure including pseudoknots computation is NP-hard[11]. However, there are available dynamic programs that find the optimal secondary structure of RNA for a subclass of pseudoknotted structures [2, 12–18]. These algorithms range from $O(n^4)$ to $O(n^6)$ asymptotic computation time

[1] A fold does not contain a pseudoknot if for $seq(1..n)$ an RNA sequence of size n the folding set for seq does not contain both (i,j) and (i',j') if $i < i' < j < j'$.

G. Lin (Ed.): COCOA 2012, LNCS 7402, pp. 176–187, 2012.

for a sequence of size n. The classification of RNA structure prediction algorithms based on the size of possible structures has been examined by Condon et al.[19] and Saule et al. [20]. The Four-Russians technique, which has been used to speedup many dynamic programs has not yet been applied to the pseudoknotted secondary structure problem.

Secondary structure algorithms for RNA do not easily lend themselves to the traditional Four-Russians technique of performing some preprocessing for a subset of all possible inputs and then computing using that preprocessing. The Four-Russians speedup technique for non-pseudoknotted RNA secondary structure, discussed in Frid and Gusfield [10], employed simultaneous and interleaved computation and preprocessing. Unfortunately, pseudoknotted secondary structure nesting is not guaranteed, thus requiring more analysis in order to execute preprocessing, and computation. Through the analysis presented here featuring *encoding* some optimal structures, preprocessing and sped up computation becomes possible. *Rivas&Eddy Algorithm* describes the broadest set of structures through its recurrences and has an asymptotic time bound of $O(n^6)$ for an RNA sequence of size n. We present *Fast R&E* algorithm which applies the Four-Russians technique to a modified maximum matching *Rivas&Eddy Algorithm*. Our approach is the first to achieve a $\Omega(log(n))$ speedup without reducing the set of possible structures.

Since the solution set of *Fast R&E* is identical to the solution set of the original *Rivas&Eddy Algorithm*, both algorithms contain equivalent limitations: the set of possible structures is restricted and the optimal structure chosen is dependent on the scoring scheme. Therefore, like in the original algorithm, there is a need to apply scoring schemes that lead to prediction of biologically accurate structures. Our approach is also compatible with Mohl et al. [21] speedup, which based on some simple pruning, in practice achieves a linear speedup of the Reeder and Giegerich Algorithm [15].

The *Fast R&E* algorithm computes pseudoknotted RNA secondary structure in $O(n^6/log(n))$ time for the standard Four-Russians speedup and in $O(n^6/log^2(n))$ time based on Pinhas et al.[22] and Williams et al. [23]. The ideas that allow for both a $log(n)$ and a $log^2(n)$ speedup are examined in the following sections.

2 The Basic Optimal Folding Problem

Let *seq* be an RNA sequence over the four-letter alphabet {A,U,C,G}, where each letter in the alphabet represents an RNA *nucleotide*. Let nucleotides x, y at position i and j in the sequence be a ***permitted pair*** of nucleotides if (x, y) or $(y, x) \in \{(\text{A,U}), (\text{C,G}), (\text{G,U})\}$. For a given sequence *seq* we define the ***folding set*** M as a set containing disjoint permitted pairs of sites in sequence *seq*. Let β be a scoring scheme such that $\beta(i, j)$ returns the contribution of pairing nucleotide x at site i with the nucleotide y at site j. The basic scoring scheme sets $\beta(i, j)$ equal to $'1'$ if (x, y) is a permitted pair with $|j - i| > d$ and set $\beta(i, j)$ to $'0'$ otherwise. Richer scoring schemes β allow more biologically

significant information to be captured by the algorithm. Let $foldScore$ be the score associated with a folding set M where $foldScore = \sum_{(i,j)\in M} \beta(i,j)$.

The *optimal folding problem*: Find the set M for which $foldScore$ is maximum under some constraints. Unconstrained, there is an exponential number of possible sets M.

3 Rivas&Eddy Algorithm

We are interested in the optimal folding problem under the constraints of the Rivas&Eddy recurrence relations. Rivas&Eddy recurrences examine the largest subset of pseudoknot structures for which an optimal solution can be found in polynomial time. We will make use of a maximization of base pairs version of the *Rivas&Eddy Algorithm* for folding, as described by Mohl et al. [21] . That version maximizes the pair contributions to the fold instead of minimizing the energy of a fold.

3.1 Rivas&Eddy Recurrences

Let $\mathbf{S[i,j;k,l]}$, where $i < j < k < l \leq N$, contain the *foldscore* for optimal Rivas&Eddy fold for the subsequence $seq(i..l)$[2], where each nucleotide in position r such that $r \in j+1..k-1$ is unpaired. Clearly, S is a four dimensional matrix.

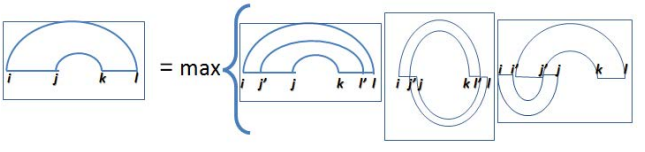

Fig. 1. Simplified S matrix recurrence as seen in Lyngso et. al. [11]

Let W[i,j] be the optimal Rivas&Eddy foldScore for subsequence $seq(i,j)$. We make use of the recurrences describing the different possible fold options, as seen below:

$$W[i,j] = \max \begin{cases} \text{Rule a.} & W[i,j-1] \\ \text{Rule b.} & W[i+1,j-1] + \beta(i,j) \\ \text{Rule c.} & \texttt{max}_{k\in\{i+1,\ldots,j-1\}} \underline{\texttt{W[i,k]}} + \texttt{W[k+1,j]} \\ \text{Rule d.} & \texttt{max}_{j',k',l'\in\{i+1,\ldots,j-1\}\wedge\{j'<k'<l'\}} \underline{\texttt{S[i,j';k',l']}} + \texttt{S[j'+1,k'-1;l'+1,j]} \end{cases} \quad (1)$$

[2] Notational note: All subsequences will be represented as seq(a..b) where a is the starting index of the subsequence and b is the index of the final character in that subsequence.

$$S[i,j;k,l] = \max \begin{cases} \text{Rule A.} & \texttt{max}\{\texttt{S[i+1,j;k,l]}, \texttt{S[i,j-1;k,l]}, \texttt{S[i,j;k+1,l]}, \texttt{S[i,j;k,l-1]}\} \\ \text{Rule B.} & \texttt{max}_{j' \in \{i+1,\ldots,j-1\}} \underline{\texttt{W[i,j']}} + \texttt{S[j'+1,j;k,l]} \\ \text{Rule C.} & \texttt{max}_{j' \in \{i+1,\ldots,j-1\}} \underline{\texttt{W[j',j]}} + \texttt{S[i,j'-1;k,l]} \\ \text{Rule D.} & \texttt{max}_{l' \in \{k+1,\ldots,l-1\}} \underline{\texttt{W[k,l']}} + \texttt{S[i,j;l'+1,l]} \\ \text{Rule E.} & \texttt{max}_{l' \in \{k+1,\ldots,l-1\}} \underline{\texttt{W[l',l]}} + \texttt{S[i,j;k,l'-1]} \\ \text{Rule F.} & \texttt{max}_{j' \in \{i+1,\ldots,j-1\} \wedge k' \in \{k+1,\ldots,l-1\}} \underline{\texttt{S[i,j';k'+1,l]}} + \texttt{S[j'+1,j;k,k']} \\ \text{Rule G.} & \texttt{max}_{j' \in \{i+1,\ldots,j-1\} \wedge k' \in \{k+1,\ldots,l-1\}} \underline{\texttt{S[i,j';k,k'-1]}} + \texttt{S[j'+1,j;k',l]} \\ \text{Rule H.} & \texttt{max}_{\{\{i',j'\} \in \{i+1,\ldots,j-1\} \wedge \{i'<j'\}} \underline{\texttt{S[i,i';j'+1,j]}} + \texttt{S[i'+1,j';k,l]} \\ \text{Rule I.} & \texttt{max}_{\{\{k',l'\} \in \{k+1,\ldots,l-1\} \wedge \{k'<l'\}} \underline{\texttt{S[k,k';l',l]}} + \texttt{S[i,j;k'+1,l'-1]} \end{cases} \tag{2}$$

Rules c, d for recurrence of matrix W and Rules B to I for the recurrence of matrix S maximize the sum of the foldScores for two sections or the RNA string. Let the section described first in each rule be called the ***head*** of the *split* and the second score corresponding to an optimal foldScore for a subsequence be called the ***tail***. For example in Rule I $\underbrace{S[k,k';l',l]}_{\textbf{head}} + \underbrace{S[i,j;k'+1,l'-1]}_{\textbf{tail}}$. We could interpret each rule as finding the optimal head and tail sum under certain conditions as seen in Figure 1. In the above recurrences Rules c,d, and Rules B to I the underlined part of each rule is the *head*, and the non-underlined is the *tail*.

3.2 Rivas&Eddy Algorithm

While many orders of evaluation are possible for these recurrences, we explicitly describe an order of evaluation that will make the Four-Russians speedup possible. Any evaluation not shown below can be done in any logical order.

The W and S matrices can be computed by an algorithm that goes through all the possible subsequences of *seq* finding the optimal Rivas&Eddy fold for each. As shown in *Rivas&Eddy Algorithm* below, the recurrences are evaluated in increasing order of the right endpoint of *seq*.

Rivas&Eddy Algorithm *Compute Matrix W::*
Initialization: **W**=0; $S(i,i;k,k)=\beta(i,k)$ or $S(i,j;k,l)=0$
for j=1 to N **do**
  for i=$j-1$ to 1 **do**
    Compute Matrix S given (i,j) (Rules A to I)
    branch_max= **compute** Rule_c(i,j) (Rule c)
    max_Rule_d = **compute** Rule_d(i,j) (Rule d)
    W[i,j]=max(*Rule a, Rule b, branch_max, max_Rule_d*)

Compute Matrix S given (i,l)*::*
for k=$l-1$ to $i+1$ **do**
  for j=$i+1$ to $k-1$ **do**
    S[i,j;k,l]=max(Rules A to I);

The score corresponding to the optimal Rivas&Eddy fold is found in $W[1,n]$. The overall asymptotic time to compute *Rivas&Eddy Algorithm* is equal to the

total time taken to compute Rules a to d for the W matrix and Rules A to I for the S matrix. The total asymptotic time results in $\{ O(n^2 * O(Rules\, a \text{ to } d)) + O(n^4 * O(RulesAtoI)) \}$ or $O(n^6)$. *Compute Matrix S give(i,j)* call is the bottleneck of the matrix W computation. For a specific i, j Compute Matrix S takes $O(n^4)$ time. Within *Compute Matrix S* the computations of Rules F-I take $O(n^2)$ time and are the bottlenecks. If you we could reduce the computation time of Rules F-I from $O(n^2)$ to $O(n^2/q)$, the overall asymptotic time of the entire algorithm would improve to $O(n^6/q)$.

We present the *Fast R&E* algorithm that, not only reduces the computation time for the bottleneck Rules F to I, but for all the Rules B to I, as well as *Rule c* and *Rule d* by a factor of q. This will lead to an overall speedup of $O(n^6/q)$ where q is $log(n)$ for the standard Four-Russians speedup. A further improvement of $log^2(n)$ maybe achieved, for the Four-Russians implementation of Pinhas et al. [22].

4 Conceptual Speedup of Rivas&Eddy

For simplicity of exposition we develop the speedup for Rule I. Rule I computes in $0(n^2)$ time and is one of the bottleneck computations. The analysis presented below leading to the speedup is then generalized to all the Rules. Given some i, j, k, l, Rule I can be computed by the following two loops:

compute Rule I for i, j, k, l::
  for $l' = l - 1$to $k + 1$ **do**
    for $k' = k + 1$ to $l' - 1$ **do**
      rule_I_max=max (*rule_I_max*, $S[k, k'; l', l]+S[i, j; k' + 1, l' - 1]$)

We conceptually divide all the possible index points of *seq* into sets of size q called **Mgroups**. Let $M_{g=0}$ be the first such group that contains the possible index points $\{0, 1, ...q - 1\}$, let $M_{g=1}$ contain $\{q, ...2q - 1\}$ and so on ... the last group of which is $M_{g=n/q} = \{n - q, ...n - 1\}$. In general:

$$M_g = \{g \cdot q, g \cdot q + 1, ...g \cdot q + q - 1\}$$

Lets define the function $\boldsymbol{K^*(RuleX, \mathbb{Y}, M_g)}$ where *Rule X* $\in$\{Rule b - d, Rule B - I\} and $\mathbb{Y}$ is a set of indexes referencing *seq* that correspond to the particular Rule X. Let $K^*(Rule\, X, \mathbb{Y}, M_g)$ return some index z, where $z \in M_g$ and maximizes Rule X under the $\mathbb{Y}$ constraint. For example:

$\mathbf{K}^*$(Rule $I, \{i, j, k, l, l'\}, \boldsymbol{M_g}$)::
  for $k' \in M_g$
    return $\boldsymbol{k'}$ such that max$\{S[k, k'; l', l]+S[i, j; k' + 1, l' - 1]\}$

Incorporating the $\boldsymbol{K^*}$-function into the computation of each Rule does not change the number of head and tail combinations examined. For example, the computation of Rule I:

compute Rule I for i, j, k, l::
  for $l' = l - 1$to $k + 1$ **do**
    for $g = \frac{k+1}{q}$ to $\frac{l'-1}{q}$ **do**
      $k' = \mathbf{K}^*$(Rule $I, \{i, j, k, l, l'\}, M_g$)
      rule_I_max=max (*rule_I_max*, $S[k, k'; l', l]+S[i, j; k'+1, l'-1]$)

The ***K****-function examines overall $O(q)$ head and tail combinations, returning the index in Mgroup M_g that results in the maximum sum. Therefore, *Rule I* compares $O(\frac{n^2}{q} \cdot O(K^*))$ head and tail combinations, totaling in $O(n^2)$ comparisons.

4.1 Breaking Up S and W into q Size Vectors

Let us conceptually break the solution matrices **S**, and **W** into vectors of size q such that for each Rule there is a vector that corresponds to the set of tails for the indexes in M_g. For Rule I let V_g be a q size vector that contains the possible tails for the indexes k' in M_g. For a particular i, j, l' and M_g, the tails examined by the ***K****-function are $\{ S[i, j; gq + 1, l' - 1]) \ldots S[i, j; gq + q, l' - 1] \}$.

Hence, $V_g = \{V_g(1) = S[i, j; \boldsymbol{gq + 1}, l' - 1], \ldots V_g(q) = S[i, j; \boldsymbol{gq + q}, l' - 1]\}$.

More precisely for *Rule I*: $V_g(m + 1 - gq) = S[i, j; m + 1, l' - 1]$.

In general for some i, j, k, l, g the vector V_g of size q indexed on $x \in \{1, \ldots, q\}$ is defined for each rule as follows:

For Rule D and I: $V_g(x) = S[i, j; gq + x, l]$;
For Rule B, F, G and H: $V_g(x) = S[gq + x; j; k, l]$;
For Rule C: $V_g(x) = S[i, gq + x - 2; k, l]$;
For Rule E: $V_g(x) = S[i, j; k, gq + x - 2]$.

We can then rewrite the computation of ***K****-function replacing the input set of indexes M_g with the vector V_g of values for the tails. We will also add the value g to the inputs, which references which Mgroup we are maximizing.

$\mathbf{K}^*(Rule\, I, \{\boxed{\boldsymbol{i, j}}, k, l, \boldsymbol{l'}\}, \boldsymbol{g}, \boldsymbol{V_g})$::
for $x = 1$ to q
  $k' = gq + x - 1$
  return $\boldsymbol{k'}$ such that max $\{S[k, k'; l', l] + \boldsymbol{V_g}(x)\}$

Note that index $\boldsymbol{i, j}$ are no longer input to the ***K****-function for *RuleI*. as the values of the scores reference by these indexes are stored in V_g

Fact 1. If x is the index point that leads to the maximum of the sum of $S[k, k'; l', l] + V_g(x)$ where $k' = gq + x - 1$ then k' is also the index point that leads to the maximum sum of $S[k, k'; l', l] + S[i, j; \boldsymbol{k' + 1}, l' - 1]$ where $k' \in M_g$.

4.2 Encoding

Optimal Rivas&Eddy scores stored in $S[i,j;\boldsymbol{k},l]$ and $S[i,j;\boldsymbol{k+1},l]$ can differ by the effect of only one more nucleotide i.e. $seq[k+1]$. Therefore, we can observe that for the scoring scheme and the recurrences of the Rivas&Eddy Algorithm, $|S[i,j;k,l]-S[i,j;k+1,l]|$ belongs to a finite set of differences $\mathbb{D}$, where $\mathbb{D}$ is the set of scores created as the result of the Scoring function β. The cardinality or size of $|\mathbb{D}|$ is $O(1)$ as a function of n. For the simple β scoring function of $+1$ for every permitted pair and 0 otherwise, the $\mathbb{D}$ set is equal to $\{0,1\}$ and therefor $|\mathbb{D}|=2$.

For the V_g vectors of Rule I the ***base*** or smallest element of the vector is $S[i,j;gq+q,l]$. Let E_g be a q size vector of differences from the *base* of V_g. For Rule I, we define $\mathbf{E_g}(\mathbf{x}) = (S[i,j;a+x,l] - \underbrace{S[i,j;a+q,l]}_{base})$, where $a=gq$.

For a particular i,j,k,l we can create and store all the E_g vectors as soon as the corresponding values in the S matrix are computed. Once computed, retrieval of any desired E_g clearly takes $O(1)$ time. The overall overhead for encoding the S matrix into a set of E matrices for the entire algorithm requires an addition of $O(n^4)$ time.

Fact 2 If x is the index point that leads to the maximum of the sum of $S[k,k';l',l]+E_g(x)$ where $k'=gq+x-1$ then x is also the index point that leads to the maximum sum of $S[k,k';l',l]+V_g(x)$. Based on Fact 1, $K^*(RuleI,\{k,l,l'\}, M_g)$ will therefore return k'.

$\mathbf{K}^*(Rule\,I,\{k,l,l'\},g,\boldsymbol{E_g})$::
for $x=1$ to q
  $k'=gq+x-1$
  return $\boldsymbol{k'}$ such that max $\{S[k,k';l',l]+\boldsymbol{E_g}(x)\}$

We can now incorporate encoded E_g vectors in the *compute Rules* functions. For example:

compute RuleI for i,j,k,l::
  for $l'=l-1$to $k+1$ **do**
    for $g=\frac{k+1}{q}$ to $\frac{l'-1}{q}$ **do**
      retrieve E_g
      $k'=\mathbf{K}^*(Rule\,I,\{k,l,l',\},g,E_g\ )$
      rule_I_max=max (*rule_I_max*, $S[k,k';l',l]+S[i,j;k'+1,l'-1]$)

For Rule I if K*()-function could be computed in $O(1)$ time, the asymptotic run-time of each rule would be reduced to $O(n^2/q)$.

Introducing table R**.** Let $\boldsymbol{R}$ be a table such that $R[X, \mathbb{Y}, g, E]$ contains the output to the $K^*(X, \mathbb{Y}, g, E_g)$-function. For example, $\mathbf{R}[I, \{k, l, l'\}, g, \boldsymbol{E_g}] = k'$ where k' is the index point that leads to the maximum of the sum of $S[k, k'; l', l] + E_g(x)$, where $x = k' - gq + 1$. Clearly,

$$\mathbf{K}^*(Rule\, I, \{k, l, l'\}, g, E_g) = \mathbf{R}[I, \{k, l, l'\}, g, E_g].$$

We, therefore, can replace all calls to function $\boldsymbol{K^*}$ with references into table $\boldsymbol{R}$.

The precomputation of table R, developed below, will allow to achieve the $\Omega(q)$ time speedup.

5 Precomputing the $\boldsymbol{R}$ Table

Assume the function call *Compute Matrix S given(1,n)* has been made, leading the variables i, l to equal: $i = 1; l = n$. During the $k = n-1$ iteration of the outer loop, the inner loop computes optimal *foldscore* for $S[\mathtt{1}, \boldsymbol{j}; \mathtt{n-1}, \mathtt{n}]$ where $j = 2, 3...$ and so on. Assume that the inner loop has completed the $j = q-1$ iteration. Hence, we have an optimal solution for $S[1, \mathbf{1}; n-1, n]$ to $S[1, \boldsymbol{q-1}; n-1, n]$. These values correspond to the *heads* of $g = 0$ for Rules d, F, G, H, I. Therefore the following algorithm precomputes $K^*(Rule\ X, \{1, n-1, n, \}, g = 0, E_{\boldsymbol{g=0}})$ for all Rules X ,for all possible E_g vectors, where $X \in \{d, F, G, H, I\}$ and stores the result in $\boldsymbol{R}$-table.

for each difference vector v of size $q-1$ such that $\forall_{1<x\leq q-1} v[x] \in \mathbb{D}$ **do**
  `compute` (an encode vector $\boldsymbol{E}$ from v) [3]
  for x=1 to q **do**
    t=$g \cdot q + x - 1$
    let max_i be the index t that makes $S[1, t; n-1, n] + \boldsymbol{E}(x)$ is maximum.
  `set` for all $X \in \{d, F, G, H, I\}$ $\boldsymbol{R}(X, \{1, n-1, n\}, g, E)$=$max\_i$

The algorithm above makes use of the fact that you can enumerate all possible sets of differences $E_{g=0}$ in $|\mathbb{D}|^q$-time.

We can generalize this algorithm for any g, as well as any i, k, l by creating an ***S_update_table*** function. Assume we have completed some M_g iteration of j (i.e. we have completed j from gq to $gq + q - 1$). We therefore have the scores for all the heads of Rules d, F-I for Mgroup M_g. Then an ***S_update_table*** function can precompute which the index in M_g would give the maximum score for every possible variation of vector E.

[3] set $\boldsymbol{E}(q) = 0$ and $\forall_{1<x\leqq -1} \boldsymbol{E}(x) = \sum_{i=q-1}^{x} v[i]$

S_update_table function (i,k,l,g)::
for each difference vector v size $q-1$ such $\forall_{1<x\leq q-1} v[x] \in \mathbb{D}$ **do**
  `compute` (an encode vector $\boldsymbol{E}$ from v)
  for x=1 to q **do**
    t=$g \cdot q + x - 1$
    $\boldsymbol{max\_i}$=t if $S[i,t;k,l] + \boldsymbol{E(x)}$ is max.
  for each Rule $X \in \{d, F, G, H, I\}$ **do**
    set $\boldsymbol{R}(X, \{i,k,l\}, g, E) = max\_i$

Compute ***fast*** *Matrix S for*(i,l)*::*
for k=$l-1$ to $i+1$ **do**
  for g=$\frac{i+1}{q}$ to $\frac{k-1}{q}$ **do**
    for $j = g \cdot q$ to $gq + q - 1 \wedge j < k$ **do**
      S[i,j;k,l]=max(Rules A-I);
    `call` *S_update_table(i,k,l,g)*

S_update_table function would be called by the *Compute fast Matrix S for (i,l)* algorithm $O(n \cdot n/q)$ times and each call would take $O(|\mathbb{D}|^q \cdot q)$ time to compute. The *Compute* ***fast*** *Matrix S for(i,l)* algorithm is presented above. While an extra loop iterated over g was added, it is clear that the call to maximize Rules A to I is made still only $O(n^2)$ times.

We would need to create a similar $W1$*_update_table* function Rules c, B, and D and $W2$*_update_table* function for Rules C, E. For example Rules d, F,G,H the precomputation occurs when the *heads* for each $Mgroup$ have their corresponding optimal solutions.

W1_update_table function(g) ::
for each vector v size $q-1$ such $\forall_{1<x\leq q-1} v[x] \in \mathbb{D}$ `compute` $\boldsymbol{E}$ from v **do**
  for $i = 0$ to $gq - 1$
    for $x = 1$ to q
      $\boldsymbol{max\_i}$=t=$gq + x - 1$ ***if*** $W[i,t] + E(x)$ is max
    for each Rule $X \in \{c, B, D\}$ *set* $\boldsymbol{R}(X, \{i\}, g, E) = max\_i$

The total asymptotic time for a single call to the *W1_update_table function(g)* function is $O(|\mathbb{D}|^q \cdot n \cdot q)$.

W2_update_table function(j,g) ::
for each vector v size $q-1$ such $\forall_{1<x\leq q-1} v[x] \in \mathbb{D}$ `compute` $\boldsymbol{E}$ from v **do**
  for $x = 1$ to q
    $\boldsymbol{max\_i}$=t=$gq + x - 1$ ***if*** $W[t,j] + E(x)$ is max
  for each Rule $X \in \{C, E\}$ *set* $\boldsymbol{R}(X, \{i\}, g, E) = max\_i$

The total asymptotic time for a single call to the *W2_update_table function(g)* function is $O(|\mathbb{D}|^q \cdot q)$.

6 Fast R&E Algorithm

Compute Matrix W::
for $g = 0$ to $\frac{N}{q}$ **do**
  for j=gq to $gq + q - 1$ **do**
    for i=$j - 1$ to 1 **do**
      Compute Matrix S for (i, j)
      branch_max= **compute** Rule_c(i, j)
      max_Rule_d = Rule_d(i, j)
      W[i,j]=`max`(*Rule a, Rule b, branch_max, max_Rule_d*)
      if $(i\%q == 0)$ *W2_update_table(j,g)* (Updating Table R for Rules C, E)
  W1_update_table(g) (Updating Table R for Rules c, B, D)

6.1 Asymptotic Analysis of Fast R&E Algorithm

For a particular g the *Fast R&E* calls the *W1_update_function* O(1) times. During one iteration of g the *W2_update_function* is called O(n) times. The total overhead for precomputing Rules c, B, D and Rules C, E is $0((n/q) \cdot [n|\mathbb{D}|^q q + |\mathbb{D}|^q nq])$ time, or simplified $O(n^2 * |\mathbb{D}|^q)$ time. For a particular i, j algorithm computes Rules A to I with function *Compute fast Matrix S* in $O(n^4/q)$ time and calls the *S_update_function* to precompute Rules d, F, G, H and I $O(n^2/q)$ times. In total, asymptotic run-time for all calls made to *Compute Matrix S* is $O(n^6/q) + O(n^4 * |\mathbb{D}|^q)$.

Finally for a particular i, j *Rule_c*(i, j) is computed in $O(n/q)$ time, *Rule_d* is computed in $O(n^3/q)$ time. The total asymptotic time for the entire *Fast R&E Algorithm* is $\underbrace{O(n^6/q + n^3/q + n^5/q))}_{computation} + \underbrace{O(n^2|\mathbb{D}|^q) + O(n^4|\mathbb{D}|^q)}_{preprocessing}$

If q=$\log_b(n)$ where the log base b is constrained by $|\mathbb{D}| < b < N$ then the asymptotic run-time is $O(n^6/log(n))$.

Memory: The original *Rivas&Eddy Algorithm* requires $O(n^4)$-space. For simplicity of exposition we chose to describe the speedup using the preprocessing of all possible tails and that requires a factor of $O(|\mathbb{D}|^q)$-space for table R. When preprocessing both heads and tails table R requires $O((|\mathbb{D}|^{2q}))$- space in total. If preprocessing all possible heads for a specific tail there is $O(|\mathbb{D}|^q)$-space requirement for table R.

Empirical Results: We ran empirical tests comparing *Fast R&E Algorithm* to the *Rivas&Eddy Algorithm* for sequences ranging in length from 100-225 nucleotides. We used the 0,1 scoring scheme setting $|\mathbb{D}|$=2. The average times for 15 sequences of each length are reported below. The standard deviation for all tests is within 5 seconds. The theoretical speedup for a sequence of 100 nucleotides is 6.64 ($log_{2.001}(100) = 6.64$) we achieved 2.33 time improvement. For a sequence size 225 nucleotides we achieved a 2.28 improvement compared to the theoretical 7.96 potential speedup.

size(n)	q	Rivas&Eddy run-time(seconds)	Fast R&E (seconds)	ratio
100	3	776.0	333.30	2.32
150	3	3,915.0	2,178.1	1.79
200	3	20,544.0	9,493.33	2.16
225	3	40,687.31	17,782.04	2.28

Conclusion and Future Work. We presented the Four-Russians speedup of $\Omega(log(n))$ for the algorithm that examines broadest set of polynomial time computed pseudoknotted secondary structures - the *Rivas&Edddy Algorithm*. The analysis explored here could be used on other pseudoknotted algorithms [12–18]. Because the solution matrices of *Fast R&E* is the same as the solution matrices of the original *Rivas&Eddy* this algorithm could be used in conjunction with other heuristic speedups [21]. It is also interesting to note that the preprocessing done here takes at most $O(n^5/log(n))$ leading to the question of whether through further preprocessing an even great speedup could be achieved.

Acknowledgments. This research was partially supported by grant IIS-0803564 from the National Science Foundation.

References

1. Condon, A., Jabbari, H.: Computational prediction of nucleic acid secondary structure: Methods, applications, and challenges. Theor. Comput. Sci. 410(4-5), 294–301 (2009)
2. Rivas, E., Eddy, S.R.: A dynamic programming algorithm for RNA structure prediction including pseudoknots. Journal of Molecular Biology 285(5), 2053–2068 (1999)
3. Torarinsson, E., Havgaard, J.H., Gorodkin, J.: Multiple structural alignment and clustering of RNA sequences. Bioinformatics 23(8), 926–932 (2007)
4. Rose, D., Hackermuller, J., Washietl, S., Reiche, K., Hertel, J., FindeiSZ, S., Stadler, P., Prohaska, S.: Computational rnomics of drosophilids. BMC Genomics 8(1), 406 (2007)
5. Torarinsson, E., Yao, Z., Wiklund, E.D., Bramsen, J.B., Hansen, C., Kjems, J., Tommerup, N., Ruzzo, W.L., Gorodkin, J.: Comparative genomics beyond sequence-based alignments: RNA structures in the encode regions. Genome Res. 18(2), 242–251 (2008)
6. Liu, C., Song, Y., Shapiro, L.: RNA Folding Including Pseudoknots: A New Parameterized Algorithm and Improved Upper Bound. In: Giancarlo, R., Hannenhalli, S. (eds.) WABI 2007. LNCS (LNBI), vol. 4645, pp. 310–322. Springer, Heidelberg (2007)
7. Nussinov, R., Pieczenik, G., Griggs, J.R., Kleitman, D.J.: Algorithms for loop matchings. SIAM Journal on Applied Mathematics 35(1), 68–82 (1978)
8. Zuker, M., Sankoff, D.: RNA secondary structures and their prediction. Bulletin of Mathematical Biology 46(4), 591–621 (1984)
9. Waterman, M.S., Smith, T.F.: RNA secondary structure: A complete mathematical analysis. Math. Biosc. 42, 257–266 (1978)

10. 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: Salzberg, S.L., Warnow, T. (eds.) WABI 2009. LNCS, vol. 5724, pp. 97–107. Springer, Heidelberg (2009)
11. Lyngsø, R.B., Pedersen, C.N.S.: RNA pseudoknot prediction in energy-based models. Journal of Computational Biology 7(3-4), 409–427 (2000)
12. Akutsu, T.: Dynamic programming algorithms for RNA secondary structure prediction with pseudoknots. Discrete Applied Mathematics 104(1-3), 45–62 (2000)
13. Dirks, R.M., Pierce, N.A.: A partition function algorithm for nucleic acid secondary structure including pseudoknots. Journal of Computational Chemistry 24(13), 1664–1677 (2003)
14. Mathews, D.H., Turner, D.H.: Prediction of RNA secondary structure by free energy minimization. Current Opinion in Structural Biology 16(3), 270–278 (2006); Nucleic acids/Sequences and topology - Anna Marie Pyle and Jonathan Widom/Nick V Grishin and Sarah A Teichmann
15. Reeder, J., Giegerich, R.: Design, implementation and evaluation of a practical pseudoknot folding algorithm based on thermodynamics. BMC Bioinformatics 5(1), 104 (2004)
16. Uemura, Y., Hasegawa, A., Kobayashi, S., Yokomori, T.: Tree adjoining grammars for RNA structure prediction. Theoretical Computer Science 210(2), 277–303 (1999)
17. Deogun, J.S., Donts, R., Komina, O., Ma, F.: RNA secondary structure prediction with simple pseudoknots. In: Chen, Y.-P.P. (ed.) APBC. CRPIT, vol. 29, pp. 239–246. Australian Computer Society (2004)
18. Cao, S., Chen, S.-J.: Predicting structures and stabilities for h-type pseudoknots with interhelix loops. RNA 15(4), 696–706 (2009)
19. Condon, A., Davy, B., Rastegari, B., Zhao, S., Tarrant, F.: Classifying RNA pseudoknotted structures. Theoretical Computer Science 320(1), 35–50 (2004)
20. Saule, C., Régnier, J.-M.S.M., Denise, A.: Counting RNA pseudoknotted structures. Journal of Computational Biology 18(10), 1339–1351 (2011)
21. Möhl, M., Salari, R., Will, S., Backofen, R., Sahinalp, S.C.: Sparsification of RNA Structure Prediction Including Pseudoknots. In: Moulton, V., Singh, M. (eds.) WABI 2010. LNCS, vol. 6293, pp. 40–51. Springer, Heidelberg (2010)
22. Pinhas, T., Tsur, D., Zakov, S., Ziv-Ukelson, M.: Edit Distance with Duplications and Contractions Revisited. In: Giancarlo, R., Manzini, G. (eds.) CPM 2011. LNCS, vol. 6661, pp. 441–454. Springer, Heidelberg (2011)
23. Williams, R.: Matrix-vector multiplication in sub-quadratic time (some preprocessing required). In: Bansal, N., Pruhs, K., Stein, C. (eds.) SODA, pp. 995–1001. SIAM (2007)