JOURNAL OF COMPUTATIONAL BIOLOGY
Volume 14, Number 7, 2007
© Mary Ann Liebert, Inc.
Pp. 908–926
DOI: 10.1089/cmb.2007.0061

A Structure-Based Flexible Search Method for Motifs in RNA

ISANA VEKSLER-LUBLINSKY,[1] MICHAL ZIV-UKELSON,[2] DANNY BARASH,[1] and KLARA KEDEM[1]

ABSTRACT

The discovery of non-coding RNA (ncRNA) motifs and their role in regulating gene expression has recently attracted considerable attention. The goal is to discover these motifs in a sequence database. Current RNA motif search methods start from the primary sequence and only then take into account secondary structure considerations. One can think of developing a flexible structure-based motif search method that will filter datasets based on secondary structure first, while allowing extensive primary sequence factors and additional factors such as potential pseudoknots as constraints. Since different motifs vary in structure rigidity and in local sequence constraints, there is a need for algorithms and tools that can be fine-tuned according to the searched RNA motif, but differ in their approach from the RNAMotif descriptor language. We present an RNA motif search tool called STRMS (Structural RNA Motif Search), which takes as input the secondary structure of the query, including local sequence and structure constraints, and a target sequence database. It reports all occurrences of the query in the target, ranked by their similarity to the query, and produces an `html` file that displays graphical images of the predicted structures for both the query and the candidate hits. Our tool is flexible and takes into account a large number of sequence options and existence of potential pseudoknots as dictated by specific queries. Our approach combines pre-folding and an $O(mn)$ RNA pattern matching algorithm based on subtree homeomorphism for ordered, rooted trees. An $O(n^2 \log n)$ extension is described that allows the search engine to take into account the pseudoknots typical to riboswitches. We employed STRMS in search for both new and known RNA motifs (riboswitches and tRNAs) in large target databases. Our results point to a number of additional purine bacterial riboswitch candidates in newly sequenced bacteria, and demonstrate high sensitivity on known riboswitches and tRNAs. Code and data are available at *www.cs.bgu.ac.il/vaksler/STRMS*.

Key words: algorithm, energy minimization, riboswitch, RNA search, secondary structure, subtree homeomorphism, tRNA.

[1]Department of Computer Science, Ben-Gurion University, Beer Sheva, Israel.
[2]School of Computer Science, Tel-Aviv University, Tel-Aviv, Israel.

1. INTRODUCTION

THE PAST FEW YEARS have witnessed an accelerated discovery of novel noncoding RNA (ncRNA) motifs such as miRNA, snoRNA and riboswitches that fill important functions in many cellular processes, such as transcription, translation, splicing, DNA replication, RNA processing, and others (Eddy, 1999). ncRNAs have been identified in a wide variety of organisms ranging from bacteria to humans. Genome-scale small RNA databases are now being constructed. These databases maintain information about the representative sequences of families of small RNAs as well as their consensus structures, e.g., Rfam (Griffiths-Jones et al., 2003), and a motivation exists to search for ncRNAs of interest.

Standard sequence alignment programs are generally not sufficient for ncRNA searches, because ncRNA is largely characterized by long-range base pair interactions (secondary structure) and not solely by its primary sequence. Developed bioinformatics methods have addressed this problem in several ways, among them RNAMotif—a descriptor-based system in which the topology of base-paired regions is specified by the user (Macke et al., 2005), tools that are based on covariance models (Eddy and Durbin, 1994) such as RSEARCH (Klein and Eddy, 2003) and CMFinder (Yao et al., 2006), secondary structure profiles like RNAProfile (Pavesi et al., 2004), methods based on structural filters like fastR (Zhang et al., 2005), and the structure to string (STR2) approach (Bergig et al., 2004) developed in our group.

As in Bergig et al. (2004), the method we describe in this paper employs pre-folding of subsequences of the target database. Theoretically, the pre-folding could be based on any of the available structure analysis methods such as x-ray crystallography, NMR, structural predictions based on multiple alignment of homologous sequences (Hofacker et al., 2002) or alternatively on MFE-based secondary structure prediction tools. The first two methods still lack sufficient amount of data, the third is dependent on the availability of sets of target sequences with high sequence homology, and the fourth method is widely used with mfold and the Vienna package (Zuker, 2003; Hofacker, 2003). Other methods that predict secondary structure with pseudoknots were developed (Rivas and Eddy, 1999), but they are still very time consuming. Thus, in its current implementation, our engine first predicts the secondary structure of the target text in consecutive sliding windows of constant size, using either mfold (Zuker, 2003) or the Vienna package (Hofacker, 2003), where suboptimal foldings (Zuker, 1989; Wuchty, 1999) are also taken into account. However, our tool could very easily be adapted to accept as input a database of RNA secondary structures based on any of the existing structure analysis tools.

Briefly, our method consists of two phases: a preprocessing phase and a search phase (Fig. 1). In the preprocessing phase we prepare the target database for a variety of future queries. This is done by partitioning the target text into given size consecutive overlapping windows with a predefined overlap, then folding each window by mfold (Zuker, 2003). This yields the optimal, as well as a few sub-optimal structures for each window. Each of the predicted structures is then converted to its tree representation yielding the "tree data base" (TDB). During the search phase, we perform a tree alignment and filter according to our pre-defined constraints. The division into two phases enables the user to run various queries and refine the constraints of each query search without reinvesting time in folding the target database.

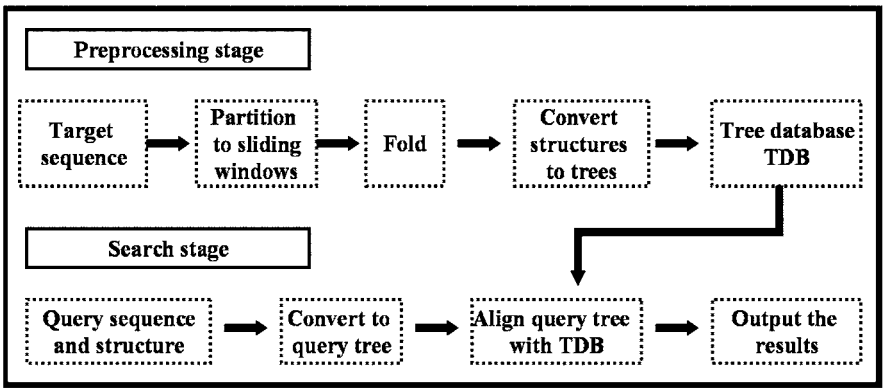

FIG. 1. Program flow.

There are several ways to represent RNA secondary structure as a graph or an ordered tree, each providing a different level of information about the structure (Shapiro, 1988; Waterman, 1978; Zhang, 1998; Le et al., 1989; Fontana et al., 1993; Steffen et al., 2006; Liu et al., 2005). General graph representation of RNAs began with the seminal work of Waterman (1978). In this model the secondary structure is a graph on the set of n labeled points (the nucleotides) and two kinds of edges connect them (adjacent nucleotides in the primary sequence and nucleotides forming base-pairs). Shapiro (1988) represented the RNA structure as a tree, where the nodes correspond to elements of secondary structure (hairpin loop, bulge, internal loop or multi-loop) and the edges correspond to base-paired (stem) regions. In Zhang's (1998) work, the representation is different: the nodes of the tree represent either unpaired bases (leaves) or paired bases (internal nodes). Each node is labeled with a base or a pair of bases, respectively. There are two kinds of edges, alternatively connecting either consecutive stem base-pairs or a leaf base with the last base-pair in the corresponding stem.

The aforementioned trees are rooted and ordered, their order corresponds to the $5'$-$3'$ orientation of an RNA sequence. Notice that the tree representation implies that pseudoknots are not taken into account. Our tree representation is conceptually compressed as in Shapiro (1988) with the exception that we have a node for every single strand component in multiloops. Moreover it may include additional information on nodes and on edges for the purpose of sequence analysis. Thus it is more informative than Shapiro's coarse grain tree representation and more compact than Zhang's (1998). This leads to a precise screening of the target text by first selecting candidates whose structural tree representation is similar to that of the query, and then further filtering these candidates by applying sequence considerations, as described in Section 2.5.

Ordered tree comparison is generally computed by tree edit distance (Shapiro and Zhang, 1990; Hochsmann et al., 2003), which allows various forms of deletions and insertions in both query and target. We observe that the search for small non-coding RNAs naturally yields a more specific tree search formulation since we do not allow deletions in the query. In our method, we apply a weighted pattern matching algorithm for finding the best homeomorphic mapping between two rooted ordered trees. Specific constraints on the searched motif can be defined in the input to the search. These may include structural constraints, such as lengths of secondary structure elements, allowing or forbidding element deletion in the target, as well as sequence constraints, such as the existence of sibling pseudoknots (Fig. 2) or local conserved sequence segments.

Based on this method we developed STRMS. We applied our tool to known and new tRNA and riboswitch searches in genomic datasets. Our alignment model, the algorithm supporting it, and its implementation are described in Section 2. The experiments, their results, and the usage of STRMS queries are described in Section 3. We conclude in Section 4 with a brief discussion.

2. METHODS AND ALGORITHMS

2.1. The tree representation

Shapiro and Zhang (1990) noted that it would be worthwhile to look into more detailed description of multiloops (Sankoff et al., 1983). We followed this recommendation and extended Shapiro's tree representation (Shapiro, 1988), while adding also sequence information. Our tree representation consists of four types of nodes and two types of edges (Fig. 2). A small circle represents the origin of a single structure (the root of the tree or a multi-branching point). A large circle represents a hairpin loop. A rhombus represents a degree-2 node such as a bulge loop, where the rhombus is colored on the side of the bulge, or an interior loop where the rhombus is colored on both sides. A large square represents all the other single-strand components that appear in the multiloops (junctions) or dangling ends.

Each stem is represented by an edge (dashed in Fig. 2), and a large square node is connected by an edge (solid in Fig. 2) to the multi-branching node or to the root. All the single-strand components (large square, large circle, and rhombus) are annotated with the length (number of nucleotides) and sequence of their respective components. A small circle node carries only topological information, and edges which correspond to stems are annotated with their respective lengths and sequences.

The information required to generate the tree structure is defined in a file (which is usually the output of the folding algorithm and contains the base-pair information). The tree construction is ordered by the $5'$

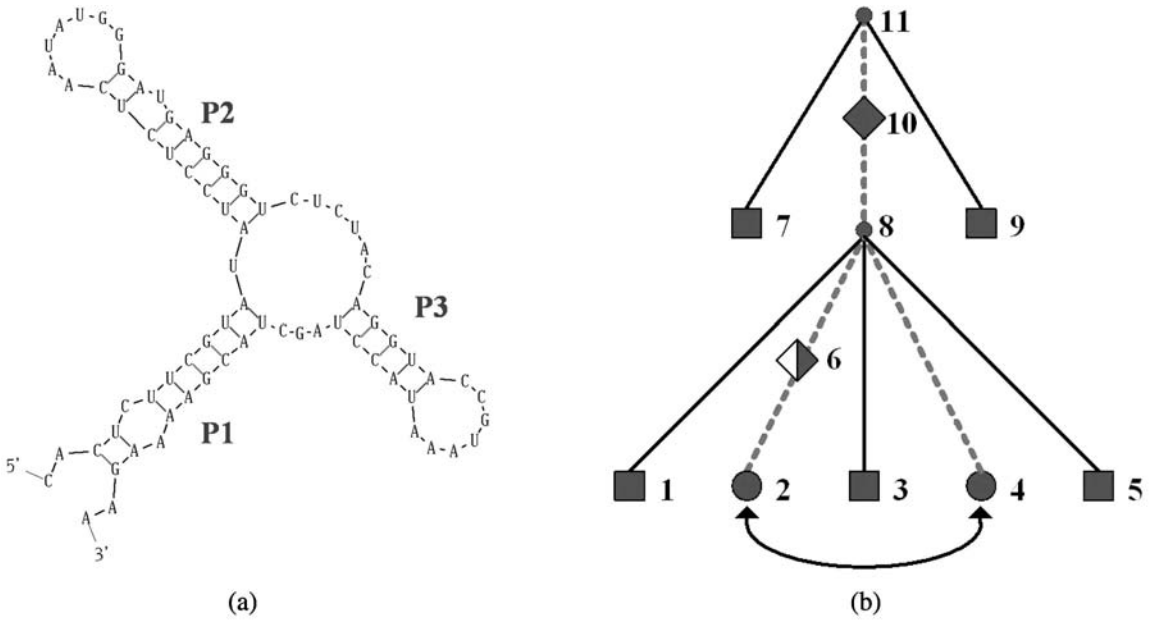

FIG. 2. A tree illustration. **(a)** Riboswitch with accession number AP001509.1. **(b)** Corresponding tree representation. In more detail, (1) 6,10 are 2-degree-nodes; 2,4 are sibling loops that form sibling pseudoknot; 8,11 are origin of a structure nodes; 10 is an internal loop; 7,9 are dangling ends; 1,3,5 are single strand components of the multiloop; 6 is a bulge; 2,4 are hairpin loops. (2) The dashed edges correspond to stems. (3) The ordering is such that the structural elements that are closest to the $5'$ end of the molecule appear on the left side of this representation, and the numbering order is bottom up, from $5'$ to $3'$.

to $3'$ ordering of the molecule. The original sequence data has thus been compressed to a concise structure form which retains also the sequence information.

2.2. The weighted subtree homeomorphism

Our search is based on a subtree homeomorphism algorithm for ordered trees, which is a relaxed version of subtree isomorphism. The *subtree isomorphism problem* (Matula, 1968, 1978) is: given a pattern tree P and a text tree T, find a subtree of T which is isomorphic to P, i.e., find if some subtree of T that is identical in structure to P can be obtained by removing entire subtrees of T, or decide that there is no such tree. The *subtree homeomorphism problem* (Chung, 1987; Reyner, 1977; Pinter et al., 2004) is a variant of the former problem, where degree-2 nodes can be deleted from the text tree (Fig. 3).

Biologically, when aligning two stems, point-mutation events could easily result in an extra bulge in an RNA structure. However, in some cases the functional homology to the original, non-mutated structure is

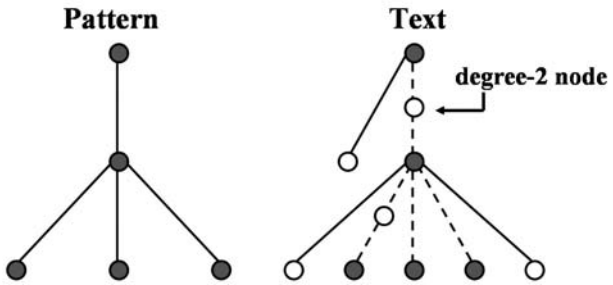

FIG. 3. A general example of subtree homeomorphism. The nodes of the text subtree that is homeomorphic to the pattern tree are filled and the edges are dashed.

still preserved. For example, the riboswitch in Figure 2a has a bulge in P2, while its functional homologue in Figure 5a does not have one.

Translating this biological description into tree comparison terms implies that the suggested alignment should be flexible enough to allow the deletion of degree-2 nodes from the target tree. Furthermore, in some cases subtrees may be deleted from the target tree but not from the query tree, as demonstrated by the tRNA search example in Figures 8 and 9 below. Both of the above application-specific properties are captured by subtree homeomorphism. Subtree homeomorphism on ordered rooted trees is more efficient (quadratic in input size) than tree edit distance (cubic in input size). Thus, by utilizing the biological properties that are typical to our application we obtain a fast variant of weighted subtree-homeomorphism on ordered rooted trees that captures our search criteria.

We start by defining a subtree homeomorphism score. Let T_1 and T_2 be two ordered, rooted, homeomorphic trees, and let $x \in T_1$, $y \in T_2$ be corresponding nodes. We define $C\_v(x, y)$ to be a user defined node-to-node similarity score function which takes into account, e.g., length differences, and $C\_e(e_1, e_2)$ to be the edge-to-edge similarity score function where $e_1 \in T_1$, $e_2 \in T_2$ are corresponding edges. Let $\delta_1(y)$ and $\delta_2(y)$ be the (usually negative) scores for deleting a node from T_2 (where δ_1 denotes the penalty of deleting a degree-2-node and δ_2 is the penalty for deleting any other node). A *mapping* $\mu : T_1 \to T_2$ is a one-to-one map from the nodes of T_1 to the nodes of T_2 that preserves the ancestor relations of the nodes and their relative order. The *subtree homeomorphism score* of the mapping, denoted $S(\mu)$, is

$$S(\mu) = \sum C\_v(u, v) + \sum C\_e(e_u, e_v) + \sum \delta_1(y) + \sum \delta_2(y) \tag{1}$$

over all $(u, v) \in \mu$, $(e_u, e_v) \in \mu$, and $y \in T_2 \setminus T_1$. Notice that $(e_u, e_v) \in \mu$ is a slight abuse of notation implicitly defined by the vertex mapping. Given two rooted ordered trees, P and T, and a score function as defined above, the *weighted subtree homeomorphism problem* is to find a homeomorphism-preserving mapping μ: $P \to t$ from P to some subtree t of T, such that $S(\mu)$ is maximal.

In our model, the cost function varies from one application to another, depending upon the amount of information supplied with the query. The simplest one just compares the topology of the structures. More complex functions include length differences of the structural elements, sequence conservation and pseudoknot matching. The node deletion score (i.e., gap penalty) reflects the tradeoff between a gap and a mismatch. As the gap penalty increases, the algorithm tends to match distant nodes to avoid gaps. As different values may suit different needs, our tool enables users to set this parameter for each run.

2.3. The tree alignment algorithm

The tree alignment algorithm employs a bottom-up two level dynamic programming (DP) and computes optimal alignments between P and any homeomorphic subtree t of T which maximizes the homeomorphism score between P and t. This approach yields an $O(mn)$ algorithm, where m and n are the number of vertices in P and T respectively (in practice, both m and n are very small in comparison to the sizes of the corresponding sequences). The bottom-up computation requires computing scores for all subtrees of P and T.

Let T be the text tree and P be the pattern tree. Let p_u denote a subtree of P rooted in node $u \in P$, and t_v denote a subtree of T rooted in node $v \in T$. Let $\langle y_1, \ldots y_{c(v)} \rangle$ be the ordered set $(5' \to 3')$ of children of node v, and $\langle x_1, \ldots x_{c(u)} \rangle$ be the ordered set $(5' \to 3')$ of children of node u, where $c(u)$ and $c(v)$ indicate the number of children of u and v respectively. We define $score(u, v)$ to be:

$$score(u, v) = \begin{cases} S(\mu) & \text{if there exists a mapping } \mu : p_u \to t_v \\ -\infty & \text{otherwise.} \end{cases} \tag{2}$$

Figure 10 and the pseudocode for the algorithm (both are included in the attached appendix) demonstrate the two-stage DP approach to the tree alignment. We use the term "large DP" (L_{DP}) for the top-level dynamic programming which fills an $m \times n$ table, where entry (u, v) contains $score(u, v)$ and a record of the nodes of t_v that were mapped to p_u. During the computation of each non-leaf entry (u, v) in the L_{DP}, a second-level dynamic programming stage, "small DP" (S_{DP}) is activated in order to compute the optimal mapping between the children of u and the children of v.

The computation of $score(u, v)$ is done recursively in a *postorder* traversal of T and P. First, $score(u, v)$ values are computed for all leaf nodes of T and P. Next, $score(u, v)$ values are computed for each node pair in P and T, based on the values of the previously computed scores for all children of u and v as follows: If $c(u) \leq c(v)$ (no deletions from the pattern are allowed) S_{DP} is computed for sequences $\langle x_1, \ldots, x_{c(u)} \rangle$ and $\langle y_1, \ldots, y_{c(v)} \rangle$. At this stage $score(x_i, y_j)$ for $i = 1, \ldots, c(u)$, and $j = 1, \ldots, c(v)$, have been computed and stored in L_{DP}. The cost of the diagonal edge in cell (x_i, y_j) of the S_{DP} is set to $score(x_i, y_j)$. The costs of the vertical edges are set to $-\infty$ to reflect the fact that no deletions are allowed from the pattern. All horizontal edges are assigned the cost of deleting a node from T (denoted by δ_2). Let *OptP* (see the pseudocode in the attached appendix) be the highest scoring path in S_{DP}. Then $score(u, v)$ is assigned to be:

$$score(u, v) = \max \begin{cases} C\_v(u, v) + C\_e(e_u, e_v) + \sum_{e \in OptP} cost(e) \\ \\ compareWithChild(u, v) + \delta_1(v) \end{cases} \tag{3}$$

The procedure $compareWithChild(u, v)$ returns $score(u, y)$ if v is a degree-2 node and y is its only child, and $-\infty$ otherwise. It is easy to see that Equation (3) recursively optimizes Equation (1). The algorithm returns a vertex $v^* \in T$ that maximizes the score $S(\mu : P \to t_{v^*})$ (found in the last row of L_{DP}).

2.3.1. Time complexity analysis. Let W denote the sliding window size and N the size of the target sequence. Let m denote the number of nodes in the tree representation of the query and n denote the number of nodes in the tree representation of a folded window (of the target sequence). The algorithm consists of an off-line preprocessing stage executed only once, followed by runs of a search stage as the user finds necessary. Therefore, we consider the main stage (search) to be the practical online bottleneck to the efficiency of our alignments.

The off-line preprocessing stage. Folding each window, during the preprocessing stage, can be done in $O(W^3)$. This yields $O(NW^3)$ when summing over all N windows. Constructing each tree from its folded representation is an additional $O(W)$ for scanning the output of the folding program (e.g., the *ct* file). Altogether, this yields an additional $O(NW)$ work for tree construction, so the total time complexity of the preprocessing stage is $O(NW^3)$.

The search stage. For each given query, the search stage iterates over all $O(N)$ trees in the TDB and applies the subtree homeomorphism algorithm to each query-text tree pair. The algorithm computes an $O(mn)$ dynamic programming matrix, denoted L_{DP}. For each computed entry (u, v) in the L_{DP} matrix, the core work is that of computing the corresponding S_{DP} dynamic programming matrix in $O(c(u) \cdot c(v))$.

We utilize the following observation in the time complexity analysis.

Observation 1. $\sum_{u=1}^{m} c(u) = m - 1$ *and* $\sum_{v=1}^{n} c(v) = n - 1$.

Summing the work over all nodes in the *Large DP* matrix, we get

$$O\left(\sum_{u=1}^{m}\sum_{v=1}^{n}(c(u)c(v))\right) \overset{\text{Obs. 1}}{=} O(mn).$$

2.4. Dealing with potential pseudoknots

In this section, we extend the subtree homeomorphism algorithm for ordered rooted trees, described in Section 2.3, to handle the pseudoknot considerations posed by the riboswitches in our study. Consider the riboswitch in Figure 2a and its corresponding tree representation in Figure 2b. Node 2 in the tree represents loop number 2 and node 4 represents loop number 4. Note that the subsequence "GGUAU" in loop 2 is complementary, according to RNA base-pairing rules, to the subsequence "CCGTA" in loop 4, and this could lead to the formation of a pseudoknot. Indeed, Mandal et al. (2003) predicted a potential pseudoknot between the two arms of the purine riboswitch aptamer. In order to extend our model to take such key information into consideration we annotate the tree in Figure 2b with this additional information by connecting node 2 and node 4 with a "potential pseudoknot" edge.

In general, these edges break down the tree-like representation of the RNA secondary structures, and thus the subtree homeomorphism algorithm may no longer apply. However, note that the potential pseudoknot in the riboswitch query in Figure 2 is confined to the subtree rooted in node 8, i.e., node 2 and node 4 are sibling nodes sharing a common parent node. More generally, we observe that, for all riboswitch aptamer queries in this study, only one potential pseudoknot is predicted, and furthermore, such a pseudoknot is always formed between two sibling leaf nodes sharing a common parent node. Thus, we use the term *sibling pseudoknot edges* to denote pointers that connect such potential pseudoknots in the query.

We extend the subtree homeomorphism algorithm to take query-to-text sibling pseudoknot matching into account, if such is specified by the user, as part of the search input profile. The text subtrees, however, could be annotated with any number of potential sibling pseudoknots, based on loop sequence complementarity analysis that is executed in the preprocessing stage (immediately after the folding and prior to the tree construction). Therefore, let Q_v denote the set of distinct potential pseudoknots in t_v and q_v denote the size of Q_v.

2.4.1. A high-level overview of the enhancement with pseudoknot matching. We start by noting that the L_{DP} stage remains unaffected by the pseudoknot matching enhancement. This is due to the fact that, by definition, each pseudoknot edge is confined to a single query subtree p_u and thus will only be matched to corresponding edges in a subtree t_v during the call to procedure *compareSubtrees*(u, v) for the computation of entry (u, v) in the L_{DP}. Therefore, it remains to show how the S_{DP} needs to be updated to enforce the sibling pseudoknot edge matching. For this, consider the dynamic programming graph in Figure 4. We annotate the S_{DP} table with arcs to represent the pseudoknots: pseudoknot X in the query, and candidate pseudoknots Y and Z in the text. If arc X is to be matched to arc Y, then the optimal DP path must enter block G_2 through vertex $(0, 2)$ and leave it through vertex $(3, 6)$. In this case, the weight of the optimal path will be the sum of its three components: the optimal path from vertex $(0, 0)$ to vertex $(2, 2)$ in block G_1, the optimal path from vertex $(0, 2)$ to vertex $(3, 6)$ in block G_2 and the optimal path from

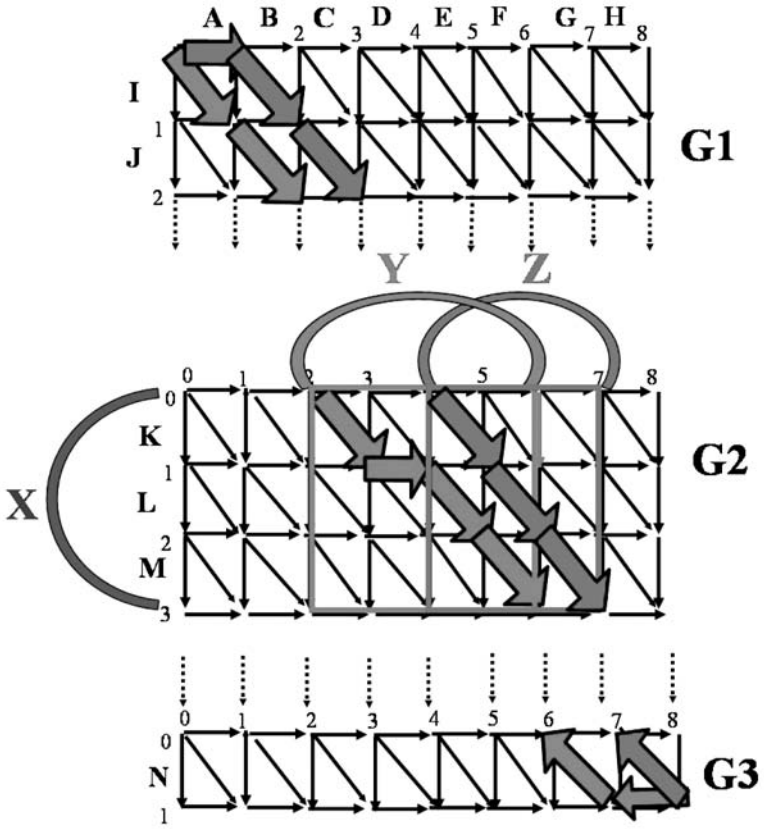

FIG. 4. The small DP stage for subtree alignment with sibling pseudoknots.

vertex $(1, 8)$ to vertex $(0, 6)$ in block G_3. Similarly, if arc Z is to be selected as the matching arc, then the optimal path must enter block G_2 through vertex $(0, 4)$ and leave it through vertex $(3, 7)$.

Following the above example, the S_{DP} is conducted as follows. Let $start_W$ denote the index of the column in G_2 where the arc begins and end_W denote the index of the column where the arc ends. For each arc W in Q_v, the optimal path that complies with the matching of candidate arc W in the text with arc X in the query is computed as the sum of three paths: the highest scoring path starting at origin vertex $(0, 0)$ of G_1 and ending in vertex $start_W$ in the last row of G_1, the highest scoring path from vertex $start_W$ in the first row of G_2 to vertex end_W in the last row of G_2, and the highest scoring path from vertex $c(v)$ in the last row of G_3 to end_W in the first row of G_3.

After the highest scoring paths corresponding to the matching of each of the candidate arcs in the text have been computed, the optimal pseudoknot matching corresponds to the highest scoring path among the q_v optional paths. It is easy to see that, when q_v is constant, the pseudoknot matching described above increases the time complexity of the main stage by a constant factor only. This is, in practice, the observed case for the riboswitch searches applied in this study, and can indeed be handled without increasing the $O(mn)$ time complexity of the search, simply by applying standard dynamic programming to compute the corresponding three paths described above for each of the candidate arcs in Q_v, and then computing the path representing the optimal matching as the maximum among $q_v = O(1)$ sums of triplets. In the next section we extend the solution to the more general case in which q_v can not be assumed to be constant.

2.4.2. The formal algorithm for enhancement with pseudoknot matching. In this section we give a formal outline of the algorithm for enhancing the S_{DP} stage to enforce the sibling pseudoknot edge matching. We will utilize the following definition of optimal paths.

Definition 1. *For any dynamic programming graph G_z, let $OPT_{G_z}[(x_1, y_1), (x_2, y_2)]$ denote the weight of the highest scoring path from vertex $(x_1, y_1) \in G_z$ to vertex $(x_2, y_2) \in G_z$.*

Recall that $\langle X = x_1, \ldots x_{c(u)} \rangle$ is the ordered set $(5' \rightarrow 3')$ of children of node u, and $\langle Y = y_1, \ldots y_{c(v)} \rangle$ is the ordered set $(5' \rightarrow 3')$ of children of node v, where $c(u)$ and $c(v)$ indicate the number of children of u and v, respectively.

The algorithm employs the following four-step approach:

Step 1. Compute the values $OPT_{G_1}[(0, 0), (i_1, \ell)]$, for $\ell = i_1 \ldots c(v)$, where i_1 denotes the number of rows in G_1. This is achieved via standard dynamic programming on the alignment graph G_1.

Step 2. Compute a data structure which encodes the values $OPT_{G_2}[(0, \ell), (i_2, k)]$, for $\ell = i_1 \ldots c(v)$ and $k = i_1 + i_2 \ldots c(v)$, where i_2 denotes the number of rows in G_2.

This can be done via all-substring-alignment techniques (Apostolico et al., 1990; Schmidt, 1998).

Step 3. Compute the values $OPT_{G_3}[(0, \ell), (i_3, c(v))]$, for $\ell = i_1 + i_2 \ldots c(v)$, where i_3 denotes the number of rows in G_3.

This can be achieved by reversing the edge directions in G_3 and running the alignment from source $(i_3, c(v))$ up to the first row of G_3.

Step 4. The best alignment between the children of u $(X = \langle x_1, \ldots x_{c(u)} \rangle)$ and the children of v $(Y = \langle y_1, \ldots y_{c(v)} \rangle)$ is computed as a maximum among q_v optimal path-score sums of triplets, where each sum represents an alignment based on a pre-conditioned matching between the only pseudoknot in p_u and a distinct candidate potential pseudoknot in t_v,

$$OPT_{G_{total}}[(0, 0), (c(u), c(v))] = \max\{OPT_{G_1}[(0, 0), (i_1, z)] + OPT_{G_2}[(0, z), (i_2, w)] + OPTG_3[(0, w), (i_3, c(v))]\}$$

$$\forall (z, w) \in Q_v.$$

2.4.3. Time complexity analysis of the enhancement with pseudoknot matching. In addition to the previous Observation 1, we utilize the following observation in the time complexity analysis.

Observation 2. $\sum_{v=1}^{n} q_v = Q \leq n^2$. *(Where Q denotes the set of distinct potential pseudoknots in T.)*

The off-line preprocessing stage. The set of potential pseudoknots is generated during the text-tree construction in the preprocessing stage by computing complementary alignments of all pairs of the exposed sequences in the hairpin loops and then storing in Q_v alignment pairs that pass a given, user-defined threshold. This is $O(W^2)$ per sliding window. This term is negligible in light of the $O(W^3)$ work invested, per window, in the folding of each subsequence during the preprocessing stage (for time complexity analysis, see Section 2.3.1).

The search stage. The subtree homeomorphism algorithm employs the above four-step computation, which is the bottleneck of the Small DP, once for each node pair $(v \in T, u \in P)$.

Steps 1 and 3 of the above procedure can be implemented in $O(c(v) \cdot c(u))$ via standard dynamic programming (the general case). Step 4 computes the maximum among q_v sums of triplets in $O(q_v)$ time. In Step 2, an $O(c(v) \times c(v))$ matrix with the values of all the highest scoring paths can naively be constructed in $O(c(u)c(v)^2)$ via standard dynamic programming. Alternatively, such a matrix can more efficiently be constructed $O(c(v)^2 \log c(v))$ time by using the algorithms of Apostolico et al. (1990) and Schmidt (1998). We employ the technique of Schmidt (1998) to construct, in $O(c(u)c(v) \log c(v))$ time, a data structure that allows to query, in $O(\log c(v))$ time, the value of any highest scoring source-to-sink path in G_2.

The total time complexity is computed by summing up the work over all node pairs.

$$O\left(\sum_{u=1}^{m}\sum_{v=1}^{n}(c(u)c(v) + c(u)c(v)\log c(v) + q_v \log c(v))\right)$$

$$\overset{\text{Obs. } 1}{=} O\left(m\sum_{v=1}^{n} c(v)\log c(v) + \sum_{v=1}^{n} q_v \log c(v)\right)$$

$$\overset{\text{Obs. } 1}{=} O(mn \log n + Q \log n)$$

$$\overset{\text{Obs. } 2, m \leq n}{=} O(n^2 \log n).$$

2.5. *Taking into account sequence considerations*

We have by now filtered the target database and ended up with a very small number of candidates that match our query specifications of structure and pseudoknot constraints. We are now ready to apply sequence constraints to these candidates. In general, our tool is very flexible and can handle a large variety of sequence considerations. In cases where the conserved functionality of the corresponding components includes single-stranded RNA-RNA or RNA-protein interactions (e.g., tRNA and riboswitches), we apply the sequence alignment criterion to the single strand regions like bulges and loops. Alternatively, in cases where the conserved functionality involves double-stranded interactions (e.g., miRNA), sequence alignment scoring is applied to the compared stems. Since sequence comparisons are performed on the small number of filtered candidates, the effect of its runtime on the overall search is negligible.

2.6. *Implementation details*

STRMS is written in Java. It is run from the command line and works on a Linux operating system. The program is divided into a number of packages for different tasks, such as extracting genes/UTRs and other parts from genomes, folding a large set of sequences using mfold, aligning trees etc. The user-defined parameters for the application, such as the query constraints, are read from a file. For each query, the program reports all the matches with positive score (or a number that can be specified by the user), sorted by score, and produces an `html` file that displays graphical images of the predicted structures as well as the tree representation for both the query and the candidate hits.

3. EXPERIMENTAL RESULTS

The sequences of the organisms presented in this section were downloaded from NCBI (*www.ncbi.nlm.nih.gov/*). The data necessary for identifying genes that are related to the purine metabolic pathway, was taken from (*www.genome.jp/*).

3.1. Riboswitches

To illustrate our algorithm, we focus on the recently discovered RNA genetic control elements, called riboswitches, that were found in bacteria (Winkler and Breaker, 2003; Nudler and Mironov, 2004). In our study we focus on the purine riboswitch that binds guanine/adenine to regulate purine metabolism and transport.

The guanine/adenine riboswitch aptamer query. We used the consensus of the purine riboswitch aptamer domain (G-box) (Mandal et al., 2003) taken from Rfam as the query structure for demonstrating our method. Its secondary structure is well predicted by mfold when applied to most of the bacterial sequences appearing in Rfam. Thus, we can compare the G-box shape with secondary structures from target gene sequences by the folding prediction of these sequences.

The secondary structure of the G-box (Fig. 5a), is composed of a three-stem (P1 through P3) junction with a multiloop connecting two hairpins and the 5′-3′ end. Significant sequence conservation occurs within P1 and in the unpaired regions. Some base-pairing potential exists between the two stem-loop sequences, which might permit the formation of a pseudoknot (Mandal et al., 2003).

At the time when our study was conducted, Rfam contained 37 seed sequences for the riboswitch and an additional 63 sequences that are outside the seed. We used the 37 seed sequences to learn characteristic constraints of the structure (e.g., element length ranges) and sequence conservation, by first folding each sequence using mfold and then learning this data by automatic application. This resulted in a tree (Fig. 5b) with the constraints exemplified in Figure 5c. In the *topological constraints* we included the penalty of deleting nodes during the alignment in S_{DP}. The penalty $\delta_2(f.child)$ denotes the penalty for deleting a child of any node in the target tree, when compared to the children of f. Other penalties for deleting nodes are set to zero as a default. In the *length constraints* we include the length ranges of the stems and the loops. *Sequence constraints* contain information about the conserved regions and their location (beginning, end, or middle of the sequence). Additional data that can be supplied in this file is the locations of pseudoknots, if such exist. All datasets to be described in this section were pre-folded using mfold with 7% suboptimality.

Validation of STRMS on known G-box instances in prokaryotes. In order to test STRMS, we performed two experiments. In the first experiment we generated a random target sequence of 5000 nts and planted in it the 100 sequences of known riboswitches that we downloaded from Rfam. We applied

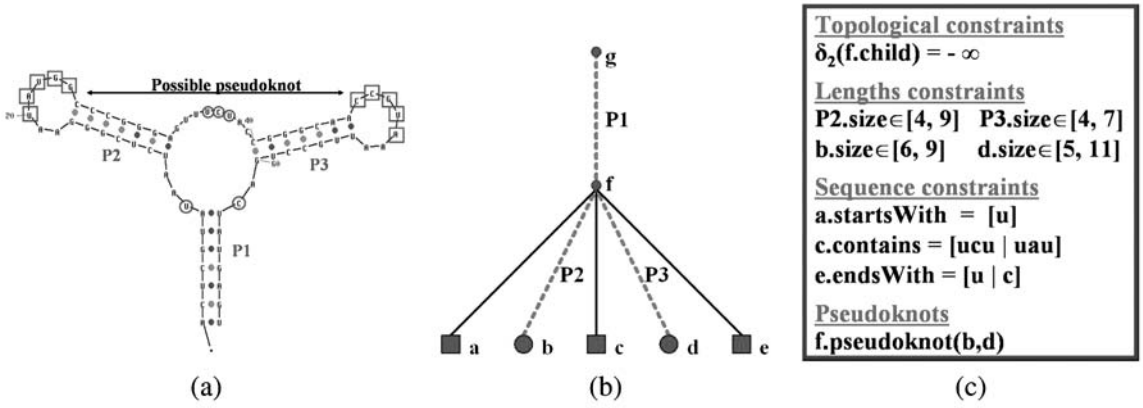

FIG. 5. A sample query for STRMS. **(a)** Folded consensus, the nucleotides marked with circles represent sequence conservation constraints and the nucleotides marked with squares represent complementarity associated with potential pseudoknots. **(b)** The corresponding representing tree. **(c)** Constraints that were imposed during this search.

TABLE 1. RESULTS OF THE RIBOSWITCH SEARCH

Organism	Number of genes	Purine genes			False positives in the control set		
		TP	***FP***	***FN***	S_1	S_2	S_3
Bacillus subtilis (NC_000964)	50	5	**0**	0	370	70	**0**
Cl. perfringens (NC_003366)	55	4	**2**	0	409	46	**1**
Bacillus anthracis (NC_003997)	44	6	**0**	0	430	47	**1**

TP are known riboswitches (found in Rfam database) that were identified by STRMS, FP are STRMS hits that were not previously reported in Rfam, and FN are known riboswitches that were missed by STRMS.

STRMS on this target sequence and found 94 of the 100 riboswitches. Checking the riboswitches that were not reported as hits revealed that the riboswitch structure did not appear among the 7% suboptimal foldings that were given as the output of mfold. We folded them again, this time relaxing the mfold threshold to generate candidate foldings with up to 15% suboptimality. The remaining 6 riboswitch structures were then found among either the 4th or the 5th suboptimal foldings.

In the second experiment, we performed a search on three organisms: *Bacillus subtilis*, *Clostridium perfringens*, and *Bacillus anthracis*. For each of the three organisms we constructed two datasets of about 50 genes each: (1) A dataset of purine metabolism genes that consists of 5′-UTRs of 500 nts each,[1] taken from upstream regions of purine metabolism genes (including the reported locations taken from Rfam); (2) a dataset of the same number of 5′-UTRs of 500 nts each, selected randomly from upstream regions of genes that do not participate in purine metabolism, to be used as a control set. The results are summarized in Table 1.

One can observe that in the three organisms our tool found all the known riboswitches. Thus the sensitivity ($\frac{TP}{TP+FN}$) is 1. The number of false positives in two of the organisms *Bacillus subtilis* and *Bacillus anthracis* is 0, thus the positive predictive value ($\frac{TP}{TP+FP}$) is also 1. In *Clostridium perfringens*, STRMS located two false positives (i.e., hits that were not previously reported in Rfam). This number of false positives is insignificant in respect to the large amount of data that was tested.[2]

On the second dataset, of non purine genes, the search was conducted in three stages according to the constraints in Figure 5c. In the first stage, the search was based only on topological similarity, as computed via subtree homeomorphism (results in column S_1). In the second stage, we enhanced the structural comparison with edge and loop length criteria (column S_2). In the final stage, we combined the sequence considerations into the search (column S_3). This reduced the number of false positives to zero or one, as observed in Table 1. This shows the importance of additional constraints supported by our tool in false positives control.

Searching for riboswitches in newly sequenced data. We used STRMS to predict new members of purine riboswitches in recently sequenced genomes of *Lactobacillus* genus of gram-positive bacteria (Table 2). For each of the organisms we extracted the 5′ UTRs (500 nts) of genes that are related to purine metabolism (about 50 genes). The search was conducted with the parameters and constraints from Figure 5c, and was run independently for each organism, over all genes in the purine gene-set. For each organism STRMS reported between 3 to 6 hits. Surprisingly, in all three species the highest scoring hit was a candidate riboswitch found at index ~150 upstream of the *xanthine phosphoribosyltransferase* gene. Some of the previously known riboswitches were also found in 5′ UTR of this gene in other organisms.

Manual inspection verified sequential conservation of nucleotides in the functionally critical positions that were listed in Mandal et al. (2003) (Fig. 6). Structural functionality was further asserted by running the Vienna Package RNAAlifold (Hofacker et al., 2002) multiple structural alignment program with the three candidate sequences as input. The resulting consensus is shown in Figure 7. Note the four pairs of compensatory mutations in each arm of the consensus predicted G-box, and the two consistent mutations

[1]Riboswitches may occur around 200–300 nt (Nudler and Mironov, 2004) upstream to the gene.

[2]Partitioning the database of 50 UTRs to pieces of 100 nts with a sliding window of 4 and about 3 suboptimal foldings yields about 15,000 structures.

TABLE 2. NEW PURINE RIBOSWITCH CANDIDATES

Organism	*Location in the genome and strand*	*Position relative to the gene*[a]
Lac. acidophilus (NC_006814)	237640-237705(-)	155 upstream
Lac. salivarius (NC_007929)	1357553-1357618(-)	147 upstream
Lac. delbrueckii (NC_008054)	251482-251547(-)	147 upstream

[a]All the members were found upstream of the gene *xanthine phosphoribosyltransferase*.

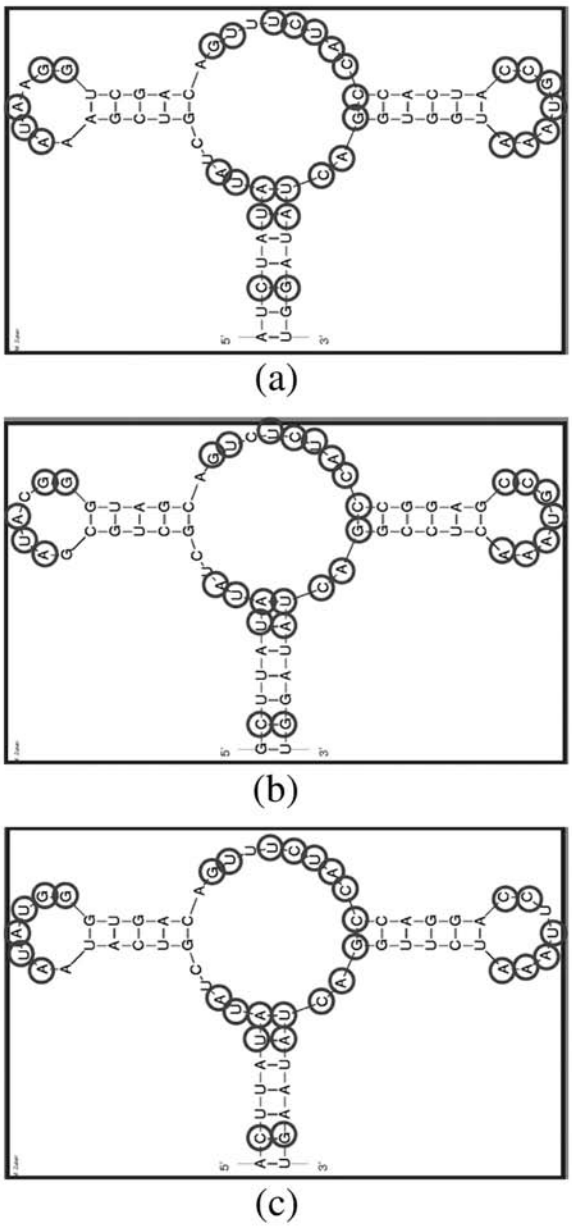

FIG. 6. Additional bacterial riboswitch candidates predicted by STRMS. The circled nucleotides comply with the conserved nucleotides that appear in the consensus structure of bacterial purine riboswitch in Mandal et al. (2003). Note that the STRMS query was based on all purine riboswitches currently in Rfam and thus had very limited sequence constraints, as specified in Figure 5c. **(a)** *Lactobacillus acidophilus* at c(237640..237705). **(b)** *Lactobacillus delbrueckii* at c(251482..251547). **(c)** *Lactobacillus salivarius* at c(1357553..1357618).

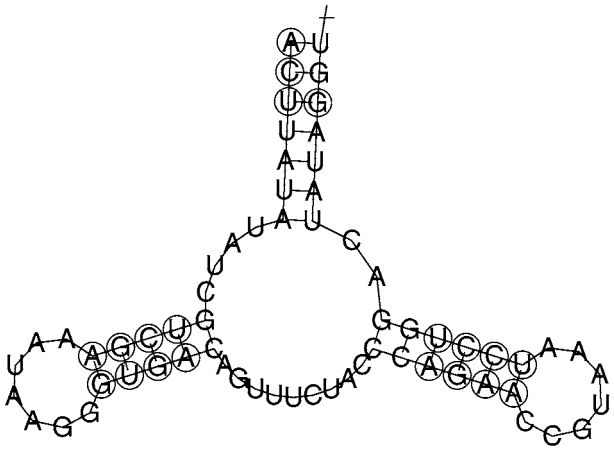

FIG. 7. The consensus secondary structure for the three candidate riboswitches that were discovered by STRMS, computed via the Vienna Package RNAAlifold. Paired positions with consistent mutations (i.e., mutations that conserve the stem structure) are indicated with circles around the varying position. Paired positions with circles around both paired bases indicate compensatory mutations (i.e., joint events where a mutation in one nucleotide was compensated by a corresponding mutation in the paired nucleotide in order to conserve the stem structure.)

and one compensatory mutation pair in the lower part of the root stem. The nucleotides in the upper part of the root stem did not show compensatory mutations, due to high sequence conservation, in agreement with the analysis in Mandal et al. (2003). This strengthens our confidence in the STRMS prediction.

Some runtime measurements. We measured the running times of STRMS on a set of 50 UTRs, each of size 500 nts, from *Bacillus subtilis* (NC_000964), on a standard PC (Pentium 4, 2.0 GHz, 512 MB RAM). The preprocessing phase was performed off-line. We partitioned the sequences to sliding windows of size 100 nts and a sliding offset of 4 nts between frames and applied mfold to get secondary structures, using 3 different levels of suboptimality. We then created the TDB. (As mentioned before, any secondary structure prediction method can be applied for this preprocessing phase.) The search phase was applied on the TDB, using the query and the constraints from Figure 5. Therefore, we measured the running times for each of the two phases separately. The results are summarized in Table 3.

3.2. tRNAs

Another test for our tool was provided by tRNA data which has been commonly used to test new methods. We extracted known tRNA genes from the tRNA database made by the program tRNAscan-SE (Lowe and Eddy, 1997).

The query. As the query we used the consensus of the tRNA genes from Rfam. The tRNA molecules range in size between 75 and 95 nts, and form a cloverleaf structure (Fig. 8a). Sometimes tRNA molecules have an extra hairpin (Fig. 9). Therefore in the search for tRNAs we allow deletions in the branching node that represents the multiloop of the tRNA. Other structure and sequence constraints are summarized in

TABLE 3. RUNTIME MEASUREMENTS OF STRMS

	Percent of suboptimality		
	5%	*7%*	*15%*
Folding time	4.2 h	5 h	10.5 h
Total number of structures	10,500	13,650	29,600
Average number of structures per piece	2.1	2.7	6
Search time[a]	9.2 min	12.5 min	26.75 min

[a]The time includes reading the tree information from file, printing the pictures of the trees that got a positive score, and producing an *html* file of the results.

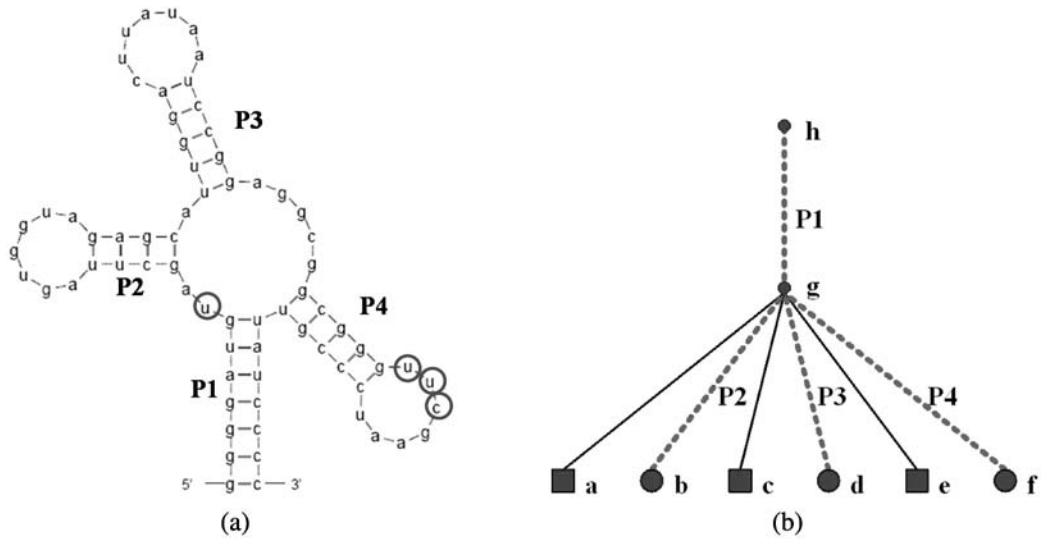

FIG. 8. Query for tRNA search. **(a)** The tRNA consensus. **(b)** The tree. The circled nucleotides in the consensus were applied as sequence conservation constraints in the search.

Table 4, for each organism separately, and were achieved similarly to that of the riboswitch (using the seed tRNA from Rfam). Figure 8 shows the predicted structure of the consensus tRNA and its representing tree.

Validation of STRMS on known tRNA instances in prokaryotes. As reported in Tsui et al. (2003), mfold predicts the cloverleaf structure as optimal folding only for 30% of the tRNA sequences. Including 7% of suboptimal foldings (averaging in 3–4 structures per sequence) increased the number of well-folded sequences to more than 60%. For each organism, we inserted randomly its tRNA sequences in a random 5000 nt sequence.

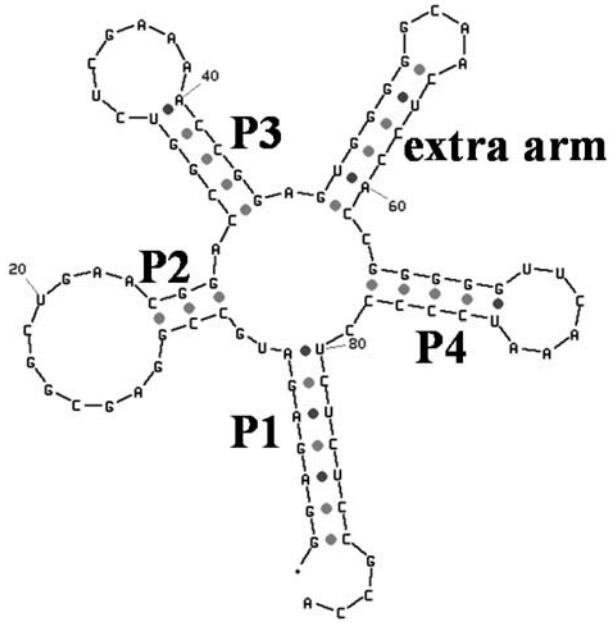

FIG. 9. An example of a tRNA from *Escherichia coli O157:H7* with an extra arm. This tRNA is located in position (2712125–2712036) of the genome. When given consensus Figure 8a as a query, our search engine is flexible enough to identify it as a matching target instance of the query. This is accommodated by the subtree homeomorphism algorithm that can be set to allow subtree deletions from the target.

TABLE 4. tRNA CONSTRAINTS THAT WERE IMPOSED DURING THE SEARCH

	Length constraints[a]						*Sequence constraints*	
Organism	*P2*	*b*	*P3*	*d*	*P4*	*f*	*a.contains*	*f.contains*
E. coli O157:H7 (NC_002695)	3–6	3–11	4–8	3–7	3–6	4–7	[U]	[UUC]
M. pneumoniae (NC_000912)	3–7	3–12	3–8	3–7	3–6	4–7	[U]	[UUC|UUU]
A. aeolicus (NC_000918)	3–6	3–11	4–8	3–7	3–7	4–8	[U]	[UUC]

[a]The location of the constraints is according to element names found in Figure 8b.

TABLE 5. STRMS RESULTS ON tRNA SEARCH

Organism	*Number*[a] *of tRNAs*	*Well*[b] *folded*	*STRMS hits*	*False positives*[c]		
				S_1	S_2	S_3
E. coli (NC_002695)	102	80	75	116	5	**0**
M. pneumoniae (NC_000912)	37	30	26	37	9	**0**
A. aeolicus (NC_000918)	44	29	27	46	9	**1**

[a]Taken from tRNAscan-SE database, including Selenocysteine tRNAs.
[b]Number of tRNAs that folded to the cloverleaf structure by mfold.
[c]Number of false positives that were reported for an arbitrary 10,000 nts sequence extracted from the organism.

As can be observed from Table 5, STRMS identifies known tRNA with a very low rate of false negatives among the well-folded sequences. For *E. coli*, it found 75 of the 80 well-folded tRNA molecules. The other 5 structures that were not found by STRMS were the ones lacking the single strand region between P2/P3 or P3/P4 (Fig. 8b). These single strand regions contain sequences that may have a complementary sequence on the other side of the stem, and the folding tool predicted them as part of the double strand. The results on *M. pneumoniae* show that 30 of 37 of the sequences folded to correct cloverleaf structures, 26 of which were found by STRMS. In *A. aeolicus*, 29 of the 44 sequences folded to correct cloverleaf structures. Among these sequences, 27 were found by STRMS. The missing structures, in both organisms, also lacked one of the single strand regions between the stems.

To test the rate of false positives that our tool reports, we extracted a piece of 10,000 nts from each genome (as a control set). We performed a three stage search in an increasing order of strictness (see columns $S_1 \ldots S_3$ of Table 5, under the title "False positives"). The first stage search (S_1) was based only on topological similarity. In the second search (S_2), we incorporated some constraints on the stem lengths and some of the unpaired regions of the multiloop (denoted as length constraints in Table 4). Finally (S_3), we combined some sequence considerations (denoted as sequence constraints in Table 4). As can be seen in these results, the number of false positives decreases dramatically when including sequence constraints in the search (S_3). These results demonstrate the flexibility of our tool and how it affects the sensitivity and of our search.

4. DISCUSSION

We devised a novel method for an emerging problem in bioinformatics, namely, the need to search for new members of known ncRNA families. Our formulation includes a score that combines both structure information and sequence constraints in a comprehensive manner. The applied combination of topological alignment with sequence comparison allows the user to restrict the sequence comparisons to specific structural elements, such as loops and stems. Our new tool is able to deal with unique queries such as those containing potential pseudoknot structure constraints (important, for example, for riboswitch searches) and

is flexible enough to accommodate the deletion of sub-structures from the target RNA during the alignment (important when searching for tRNAs, for example).

Our representation of the input query and text is very concise and thus their sizes are reduced from that of the original sequences to an efficient tree representation with m and n nodes, respectively, where m is the number of nodes in the query tree and n is the number of nodes in the text tree. We described a quadratic $O(mn)$ time RNA search algorithm based on subtree homeomorphism for ordered, rooted trees. Within this time bound we also showed a pseudoknot enhancement for the practical case of riboswitches. The more general case, when the number of potential pseudoknots is not constant, is handled in $O(n^2 \log n)$.

We employed STRMS in search for both new and known riboswitches in bacterial UTRs. Our results point to a number of riboswitch candidates in newly sequenced bacteria, and demonstrate high sensitivity on known riboswitches. We point out that, even though we demonstrated STRMS by applying it to ncRNA search, our tool is generic and could be applied to search for any RNA pattern that combines sequence and structure.

Note that our search engine leans on a pre-processing stage that predicts the secondary structure of the genomic RNA sequences in which the queries are to be sought. Thus, our approach is best-suited to applications in which the target database of folded sequences is prepared off-line and stored once, in preparation for many queries (and various constraint setting configurations per query) to follow. Alternatively, the pre-folding could be practically applied "on the fly" when dealing with focused queries. For example, one might pre-fold the set of UTRs of genes that participate in a specific metabolic pathway, as demonstrated in the Experimental Results Section, and then search the pre-folded data with specific riboswitch queries.

The pre-folding approach applied here exploits structural prediction information to its maximum, and allows extra sensitivity in cases where the folding tools correctly capture the secondary structure. We point out again that, even though in this study we applied an mfe-based tool to the pre-folding stage, our search engine is modular and the structure-prediction preprocessing stage could just as well be based on any other available RNA secondary structure prediction tool, including of course tools that combine comparative genomics with free energy minimization criteria.

Future work includes enhancements of our scores. The current score function which combines topological similarity structural constraints and sequence considerations is still in its preliminary stage. Thus, our tool currently reports all the matches above score 0, arranged by score. We are now in process of studying the statistical behavior of the reported hits in order to annotate the search results with a statistical significance value.

5. APPENDIX

Pseudocode and a running example (Fig. 10)

Procedure $score(u, v)$

Input: A dynamic programming (DP) table with all values up to cell (u, v) already set. Penalties $\delta_1(y)$ and $\delta_2(y)$, C_v and C_e functions.

Output: The score to be set to entry (u, v) of the DP table.

1. **if** u and v are leaf nodes **return** $C\_v(u, v) + C\_e(e_u, e_v)$
2. **else return**

$$\max \begin{cases} C\_v(u, v) + C\_e(e_u, e_v) + compareSubtrees(u, v) \\ compareWithChild(u, v) + \delta_1(v) \end{cases}$$

where

$$compareWithChild(u, v) = \begin{cases} score(u, y) & \text{if } v \text{ is degree-2-node,} \\ & \text{and } y \text{ is the only child of v} \\ -\infty & \text{otherwise} \end{cases}$$

(e_u, e_v are the incoming edges respectively)

Procedure *compareSubtrees*(u,v)

Input: A DP table with all values up to cell (u, v) already set.
Output: The score of aligning children of u with children of v.

$c(u)$: the out-degree of node u; $c(v)$: the out-degree of node v;
$\langle x_1, \ldots x_{c(u)} \rangle$: the set of children of u;
$\langle y_1, \ldots y_{c(v)} \rangle$: the set of children of v;

1. if $c(u) > c(v)$ return $-\infty$
2. else return $score(OptP(\langle x_1, \ldots x_{c(u)} \rangle, \langle y_1, \ldots y_{c(v)} \rangle))$

Procedure *OptP*(X,Y)

Input: $X = \langle x_1, \ldots x_{c(u)} \rangle$, $Y = \langle y_1, \ldots y_{c(v)} \rangle$
Output: The optimal alignment path between the sets.
Build a dynamic programming table $c(u) \times c(v)$.
For each cell (x_i, y_j) set three edges: vertical: $-\infty$, horizontal: $\delta_2(y_j)$ and diagonal: $score(x_i, y_j)$.
Find the optimal path using standard dynamic programming and return it.

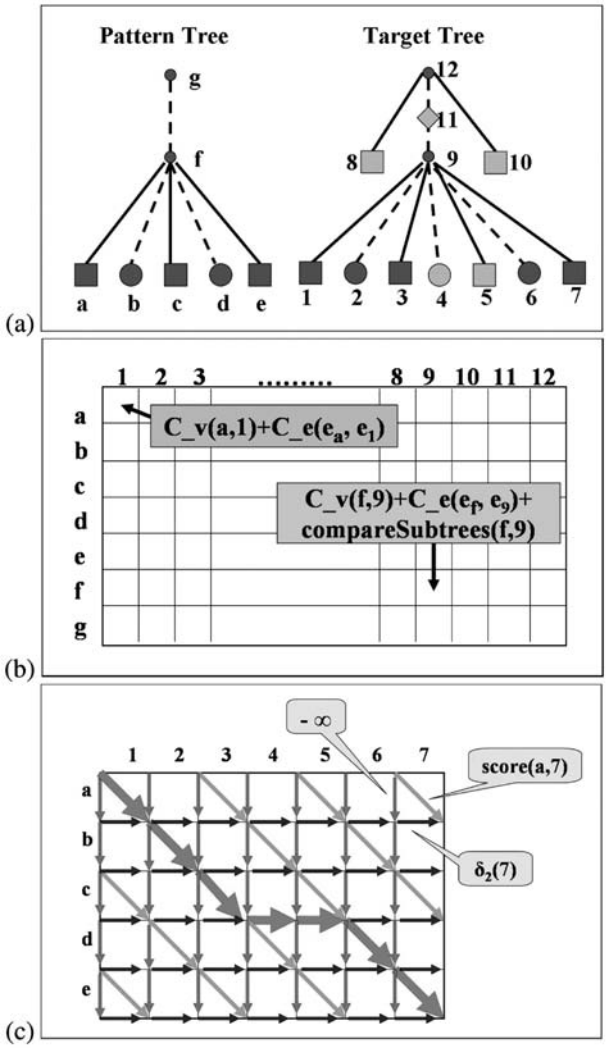

FIG. 10. A running example of the algorithm. **(a)** The compared trees. **(b)** Large DP. **(c)** Small DP—when comparing subtrees of f and 9.

ACKNOWLEDGMENTS

The research described in this paper was partially supported by a grant from the Israel USA binational foundation BSF 2003291. M.Z.-U.'s work was partially supported by an *Eshkol grant* from the Israeli Ministry of Science and Technology.

REFERENCES

Apostolico, A., Atallah, M., Larmore, L., et al. 1990. Efficient parallel algorithms for string editing problems. *SIAM J. Comput.* 19, 968–998.

Bergig, O., Barash, D., Nudler, E., et al. 2004. STR2: a structure to string approach for locating G-box riboswitch shapes in pre-selected genes. *In Silico Biol.* 4, 593–604.

Chung, M.J. 1987. $O(N^{2.5})$ time algorithms for the subgraph homeomorphism problem on trees. *J. Algorithms* 8, 106–112.

Eddy, S.R. 1999. Noncoding RNA genes. *Curr. Opin. Genet. Dev.* 9, 695–699.

Eddy, S.R., and Durbin, R. 1994. RNA sequence analysis using covarience models. *Nucleic Acids Res.* 20, 2079–2088.

Fontana, W., Konings, D.A., Stadler, P.F., et al. 1993. Statistics of RNA secondary structures. *Biopolymers* 33, 1389–1404.

Griffiths-Jones, S., Bateman, A., Marshall, M., et al. 2003. Rfam: an RNA family database. *Nucleic Acids Res.* 31, 439–441.

Hochsmann, M., Toller, T., Giegerich, R., et al. 2003. Local similarity in RNA secondary structures. *Proc. IEEE Comput. Soc. Bioinform. Conf.* 2, 159–168.

Hofacker, I.L. 2003. Vienna RNA secondary structure server. *Nucleic Acids Res.* 31, 3429–3431.

Hofacker, I.L., Fekete, M., and Stadler, P.F. 2002. Secondary structure prediction for aligned RNA sequences. *J. Mol. Biol.* 319, 1059–1066.

Klein, R.J., and Eddy, S.R. 2003. RSEARCH: finding homologs of single structured RNA sequences. *BMC Bioinform.* 4, 44.

Le, S.Y., Nussinov, R., and Maizel, J.V. 1989. Tree graphs of RNA secondary structures and their comparisons. *Comput. Biomed. Res.* 22, 461–473.

Liu, J., Wang, J.T., Hu, J., et al. 2005. A method for aligning RNA secondary structures and its application to RNA motif detection. *BMC Bioinform.* 6, 89.

Lowe, T.M., and Eddy, S.R. 1997. tRNAscan-SE: a program for improved detection of transfer RNA genes in genomic sequence. *Nucleic Acids Res.* 25, 955–964.

Macke, T.J., Ecker, D.J., Gutell, R.R., et al. 2001. RNAMotif, an RNA secondary structure definition and search algorithm. *Nucleic Acids Res.* 29, 4724–4735.

Mandal, M., Boese, B., Barrick, J.E., et al. 2003. Riboswitches control fundamental biochemical pathways in *Bacillus subtilis* and other bacteria. *Cell* 113, 577–586.

Matula, D.W. 1968. An algorithm for subtree identification. *SIAM Rev.* 10, 273–274.

Matula, D.W. 1978. Subtree isomorphism in $O(n^{5/2})$. *Ann. Discrete Math.* 2, 91–106.

Nudler, E., and Mironov, A.S. 2004. The riboswitch control of bacterial metabolism. *Trends Biochem. Sci.* 29, 11–17.

Pavesi, G., Mauri, G., Stefani, M., et al. 2004. RNAProfile: an algorithm for finding conserved secondary structure motifs in unaligned RNA sequences. *Nucleic Acids Res.* 32, 3258–3269.

Pinter, R.Y., Rokhlenko, O., Tsur, D., et al. 2004. Approximate labelled subtree homeomorphism. *Lect. Notes Comput. Sci.* 3109, 59–73.

Reyner, S.W. 1977. An analysis of a good algorithm for the subtree problems. *SIAM J. Comput.* 6, 730–732.

Rivas, E., and Eddy, S.R. 1999. A dynamic programming algorithm for RNA structure prediction including pseudoknots. *J. Mol. Biol.* 285, 2053–2068.

Sankoff, D., Kruskal, J.B., Mainville, S., et al. 1983. Fast algorithms to determine RNA secondary structures containing multiple loops, 94–120. In: Sankoff, D., and Kruskal, J.B., eds. *Time Warps, String Edits, and Macromolecules: The Theory and Practice of Sequence Comparison*. Addison-Wesley, Reading, MA.

Schmidt, J.P. 1998. All highest scoring paths in weighted grid graphs and their application to finding all approximate repeats in strings. *SIAM J. Comput* 27, 972–992.

Shapiro, B.A. 1988. An algorithm for comparing multiple RNA secondary structures. *Comput. Appl. Biosci.* 4, 387–393.

Shapiro, B.A., and Zhang, K. 1990. Comparing multiple RNA secondary structures using tree comparisons. *Comput. Appl. Biosci.* 6, 309–318.

Steffen, P., Vo, B., Rehmsmeier, M., et al. 2006. RNAshapes: an integrated RNA analysis package based on abstract shapes. *Bioinformatics* 22, 500–503.

Tsui, V., Macke, T., and Case, D.A. 2003. A novel method for finding tRNA genes. *RNA* 9, 507–517.
Waterman, M.S. 1978. Secondary structure of single-stranded nucleic acids. *Adv. Math. Suppl. Stud. 1*, 167–212.
Winkler, W., and Breaker, R.R. 2003. Genetic control by metabolite-binding riboswitches. *Chembiochem.* 4, 1024–1032.
Wuchty, S., Fontana, W., Hofacker, I.L., et al. 1999. Complete suboptimal folding of RNA and the stability of secondary structures. *Biopolymers* 49, 145–165.
Yao, Z., Weinberg, Z., and Ruzzo, W.L. 2006. Cmfinder a covariance model based RNA motif finding algorithm. *Bioinformatics* 22, 445–452.
Zhang, K. 1998. Computing similarity between RNA secondary structures. *IEEE Int. Joint Symp. Intelligence Systems*, 126–132.
Zhang, S., Haas, B., Eskin, E., et al. 2005. Searching genomes for noncoding RNA using fastR. *IEEE/ACM TCBB* 2, 366–379.
Zuker, M. 1989. On finding all suboptimal folding of an RNA molecule. *Science* 244, 48–52.
Zuker, M. 2003. Mfold web server for nucleic acid folding and hybridization prediction. *Nucleic Acids Res.* 31, 3406–3415.

Address reprint requests to:
Dr. Klara Kedem
Department of Computer Science
Ben-Gurion University
84105 Beer Sheva, Israel

E-mail: klara@cs.bgu.ac.il