Pseudoknot Identification through Learning TAG_{RNA}

Sahar Al Seesi, Sanguthevar Rajasekaran, and Reda Ammar

Computer Science and Engineering Department, University of Connecticut
{sahar,rajasek,reda}@engr.uconn.edu

Abstract. Studying the structure of RNA sequences is an important problem that helps in understanding the functional properties of RNA. Pseudoknot is one type of RNA structures that cannot be modeled with Context Free Grammars (CFG) because it exhibits crossing dependencies. Pseudoknot structures have functional importance since they appear, for example, in viral genome RNAs and ribozyme active sites. Tree Adjoining Grammars (TAG) is one example of a grammatical model that is more expressive than CFG and has the capability of dealing with crossing dependencies. In this paper, we describe a new inference algorithm for $TAG_{RNA,}$ a sub-model of TAG. We also introduce an RNA structure identification framework, TAG_{RNA}Inf, within which the TAG_{RNA} inference algorithm constitutes the core of the training phase. We present the results of using the proposed framework for identifying RNA sequences with pseudoknot structures. Our results outperform those reported in [14] for the same problem that employs a different grammatical formalism.

1 Introduction

In recent times there has been an observed acceleration in the RNA structure determination and analysis [11] owing to its paramount importance. This is partly due to the discovery of many new functional RNAs, such as miRNAs and tmRNAs [3] [16] [29]. Another factor that has led to the speeding up of RNA structural research is the rise of the RNA World Hypothesis [9] which suggests that the current DNA and protein world has evolved from an RNA based world. This Hypothesis is supported by the fact that RNA can carry genetic information like DNA and it is capable of catalyzing reactions like proteins (rRNA). Genetic information of some existent viruses is carried in RNA form [15]. Since the function of bimolecular sequences depends on its structure, analyzing RNA structures is essential to create new drugs and understand genetic diseases [6] [20]. Computational methods can provide less expensive solutions to structure analysis than other methods such as nuclear magnetic resonance and x-ray crystallography.

In the early 90's, David Searls studied the linguistics of biological sequences [23]. He suggested the use of formal grammars as a tool to model and analyze DNA, RNA, and proteins. The use of grammars has attracted the attention of many researchers [7] [26] because it can model long range interactions. In addition, grammatical models are concise and easy to understand representation of structures of sequence families. Thus, it is considered to be a natural analytical approach to fully understanding the structure and properties of these sequences. Results for secondary structure prediction

M. Chetty, A. Ngom, and S. Ahmad (Eds.): PRIB 2008, LNBI 5265, pp. 132–143, 2008.

and multiple sequence alignment agree with and sometimes suggest improvements over traditional methods [21].

Pseudoknot is one type of RNA structures that cannot be modeled with Context Free Grammars (CFG) because it exhibits crossing dependencies. Pseudoknot structures have functional importance since they appear, for example, in viral genome RNAs [15], ribozyme active sites [25], and tmRNA [28]. Among the available research in analyzing pseudoknot structures are the works of Akutsu [2], Dirks and Pierce [8], and Reeder and Giegerich [18]. These algorithms are not based on formal grammars. In the area of modeling molecular sequences grammatically, more than one model, capable of representing pseudoknots, have been presented. Cai *et. al.* [7] proposed Parallel Communicating Grammar Systems (PCGS) with an $O(n^6)$ time parsing algorithm. Another model which also requires $O(n^6)$ parsing time has been proposed by Rivas and Eddy[19]. Uemura et. al. [26] suggested the use of a submodel of TAG, TAG_{RNA}. Our solution is based on the TAG_{RNA} model.

Recently, there has been a special focus on the use of grammatical inference in bioinformatics. Sakakibara has published [22] in which he discusses the general merits of using grammatical inference in bioinformatics. Brazma *et. al.* [5] have proposed an approach to discover simple grammars for families of biological sequences. The grammatical formalisms they use are subclasses of regular patterns. On the use of grammatical inference to analyze RNA structures with Pseudoknots, Laxminarayana *et. al.* [13] presented an inference algorithm for Terminal Distinguishable Even Linear Grammars (TDELG), and they have shown how to use this algorithm in an Infer-Test model for the detection of a pseudoknot structure in an RNA sequence. The experimental results they presented [14] show 54% sensitivity when using 50% of the RNA sample for training. The sensitivity rises to 85% only when 90% of the sample is used for training. Specificity was not reported. This is the same problem as the one we address, and our results outperform those numbers, as it will be shown. Takakura *et. al.* have published [24] in which they give a linear time algorithm for generating probabilistic TAG_{RNA} from alignment data. They use the inferred grammar to find new members of nc-RNA families, which is a different problem from the one we address in this paper.

The use of grammatical inference to automate the grammar building step is essential in facilitating the use of grammatical formalism by biologists. Otherwise, the biologist will always be dependent on a grammar expert. In this work, we present a complete RNA structure identification framework, TAG_{RNA}Inf, capable of handling pseudoknot structures. By structure identification we mean, given an RNA sequence, we answer the question of whther it exhibits a certain structure or not. In our approach, the structure is represented by a TAG which is inferred from a training set. We describe a new polynomial time inference algorithm for TAG_{RNA} which constitutes the core of the training phase within the identification framework. We evaluate our solution experimentally through calculating the sensitivity and specificity of identification.

2 TAG and TAG_{RNA}

Tree Adjoining Grammars (TAGs) were originally introduced, by Joshi *et. al.* [12], for use in the field of natural language processing. Uemura *et. al.* [26] defined a subclass of TAGs, $TAG_{RNA,}$ suitable to model RNA pseudoknot structures. They developed

an $O(n^5)$ time parsing algorithm for TAG_{RNA}. Before describing TAG_{RNA} we will first give a brief introduction to the original TAG model.

A Tree Adjoining Grammar (TAG) is defined to be a 5-tuple $(T \cup \{\varepsilon\}, N, I, A, S)$, where T is a set of terminal symbols, N is a set of non-terminal symbols, ε is the empty string symbol, and S is the starting symbol. I and A are defined as follows:

- I (initial trees): A finite set of finite trees with the internal nodes' labels belonging to $N \cup \{S\}$, the leaves' labels belonging to $T \cup \{\varepsilon\}$, and the root is labeled with S.
- A (auxiliary trees): A finite set of finite trees with the internal nodes' labels belonging to $N \cup \{S\}$, and the leaves' labels belonging to $T \cup \{\varepsilon\}$ except one leaf node which has the same label as the root. This special leaf node is called a *foot node*.

Trees belonging to $I \cup A$ are called elementary trees. A tree derived by composing two other trees is called a derived tree. Trees can be composed together using the adjoining operation. The adjoining operation composes an auxiliary tree α with a foot node labeled X with any other tree β that has some internal node with the same label X. The operation works as follows: we start with the tree β and we extract the sub-tree rooted at the internal node labeled with X (let that sub-tree be γ), and replace it with the α. Then at the foot node of α, we reinsert γ. The adjoining operation is illustrated in Fig. 1. Let $T = \{ t : \exists i \in I$ s.t. t can be derived from $i\}$, then L(TAG) consists of the yield of all the trees in T.

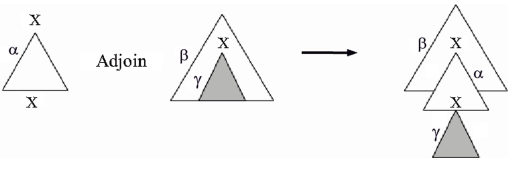

Fig. 1. The Adjoining Operation

In [26], Extended Simple Linear TAG (ESLTAG) is defined to be a subclass of TAG with adjoining constraints [27]. In ESLTAG, the adjoining operation can occur only at internal nodes tagged with the symbol *, and the number of these nodes is restricted. TAG_{RNA} is a sub-class of ESLTAG where only five types of elementary trees are allowed (Fig. 2)[1]. Each type of tree is responsible for a specific kind of branching or structural form that an RNA sequence can have.

3 The Structure Identification Framework

We introduce a complete RNA structure identification framework, TAG_{RNA}Inf, which is capable of handling pseudoknot structures. Within this framework, we present a new inference algorithm for TAG_{RNA} which constitutes the core of the training phase.

[1] Tree types of TAG_{RNA} will be explained further in section 3.1.2.

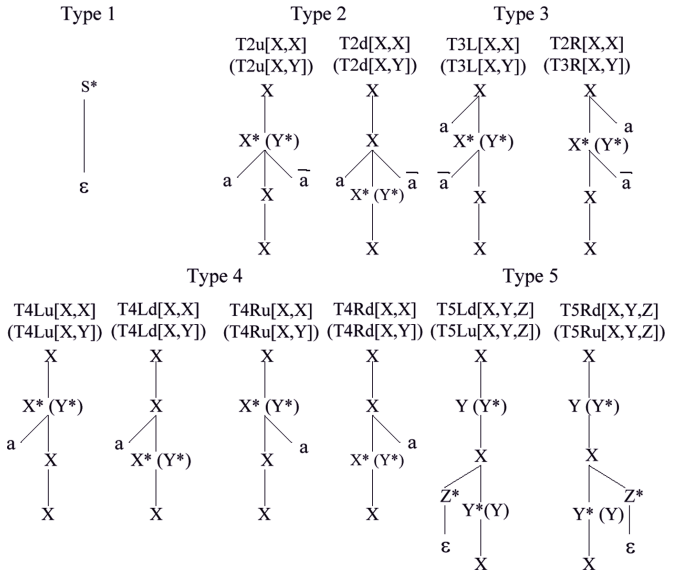

Fig. 2. TAG_{RNA}

Fig. 3 depicts the proposed framework. In the training phase, the inference algorithm is fed with a positive training set with structural information. The algorithm will generate a grammar for the provided sample. Then, the same sample along with a negative sample and the grammar generated by the inference algorithm will go through a TAG parser. For each input sequence the TAG parser will output a score. These

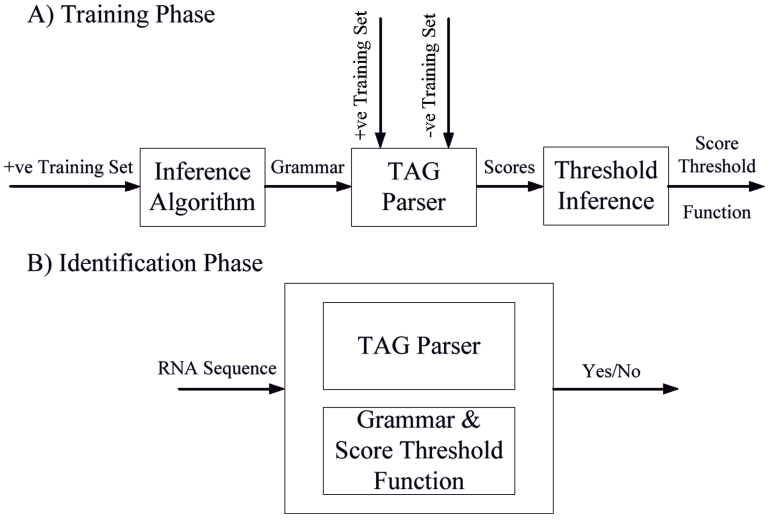

Fig. 3. TAG_{RNA}Inf : RNA Structure Identification Framework

scores will be the input to a threshold function inference module. The inferred threshold function will be used in the identification phase.

Several scoring functions can be used. For example, it can be either the number of base pairs or the minimum free energy (mfe) of the RNA sequence structure. Also, a probabilistic function can be used to generate the scores. Currently, we use the number of base pairs as the scoring function. We intend to investigate other alternatives. The inferred grammar and the scoring threshold function will be used by a TAG parser in the identification phase. Given an RNA sequence, the identification module will be able to check if this sequence has a certain structure such as a pseudoknot.

3.1 The Inference Algorithm

The grammar inference adopted here is a three step process. The input is a set of sequence data that includes the structure of each sequence, and the output is a grammar that models the input sample. If the input sample includes at least one sequence representing each RNA structure in the population being modeled, the output grammar will be a correct model for the RNA population from which the sample was drawn. For a population S, a grammar G is considered to be a correct model of S iff $S \subseteq L(G)$. For the purpose of evaluating the inferred grammar within the proposed framework, however, we calculate the sensitivity and specificity of identification.

The three steps of the inference process are: the pattern generation, the single pattern grammar generation, and the final grammar composition.

3.1.1 Pattern Generation

Definition: Let $(\chi, \overline{\chi^r})$ and $(\gamma, \overline{\gamma^r})$ be two substring pairs in a pattern p, we call the two pairs $(\chi, \overline{\chi^r})$ and $(\gamma, \overline{\gamma^r})$ a crossing dependency if $i < k < j < l$ where i, j, k, and l are the positions of χ, $\overline{\chi^r}$, γ, and $\overline{\gamma^r}$, respectively, in p.

The inputs to this phase are: the sequence size, the number of stems in the input sequence (n), the starting and ending indices of each stem in the sequence represented as a 4-tuple $(l_{i1},l_{i2},m_{i1},m_{i2})$, where if χ_i , $\overline{\chi_i^r}$ are the two strands of a stem in the sequence, l_{i1} and l_{i2} are the starting and ending positions of x_i, respectively, and m_{i1} and m_{i2} are the starting and ending positions of $\overline{\chi_i^r}$, respectively.

The pattern generation is based on sorting the pairs (l_{i1}, l_{i2}) and (m_{i1},m_{i2}) for all values of $i \leq n$ resulting in a sorted list P of $2n$ pairs (p_{i1}, p_{i2}). We maintain a link from each pair of numbers to its corresponding substring symbol χ_i's or $\overline{\chi_i^r}$'s. Thus, once the number pairs are sorted, the χ's are consequently sorted. Because any two intervals (p_{i1},p_{i2}), (p_{j1},p_{j2}) are non-overlapping we can perform the sort on the first value in the pairs, and because we are dealing with integers we can use radix sort. This will require linear time in the number of stems n. The generated pattern consists of the sorted χ's and $\overline{\chi^r}$'s with w's inserted, to represent loops in the RNA structure, wherever there is a gap between the numbers p_{i2} and $p_{(i+1)1.}$ The number of w's in a pattern must be less than or equal $2n + 1$. The insertion can be done by copying the sorted x_i's and $\overline{\chi_i^r}$'s sequentially in an array of size $4n + 1$. During the sequential copying process, we check for gaps and insert w's as necessary. This also requires linear time in n.

After the pattern is generated and before any grammar inference can be performed, we must insert the empty string symbol, ε, in the pattern. The empty string appears in TYPE 1 and TYPE 5 trees of TAG_{RNA} (see Fig. 2). Currently, we support patterns that have exactly one ε symbol. Considering all the crossing dependencies $((x, \overline{x^r}), (y, \overline{y^r},))$, ε is inserted at $i + 1$ where i is the index of the rightmost $\overline{x^r}$ in the pattern.

An Example

The pseudoknot structure at the gag-pol translational readthrough site of spleen necrosis virus [4] has the following pattern

$$w_1 x_1 w_2 x_2 w_3 x_3 \overline{x_2^r} w_4 \overline{x_1^r} w_5 \overline{x_3^r}$$

This pattern has two crossing dependencies, $((x_1, \overline{x_1^r}), (x_3, \overline{x_3^r}))$ and $((x_2, \overline{x_2^r}), (x_3, \overline{x_3^r}))$. Because $\overline{x_1^r}$ comes to the right of $\overline{x_2^r}$, the ε is inserted after $\overline{x_1^r}$.

The ε location identification is facilitated by generating a list, **links**, in which for each pair of dependent substrings $(x_i, \overline{x_i^r})$, **links**$[i] = (j, k)$ where j and k are the positions of x_i and $\overline{x_i^r}$ in the pattern, respectively. The list **links** is simply constructed by scanning the pattern once and filling the corresponding entries for each x_i and $\overline{x_i^r}$ in **links** as they are scanned in the pattern. Thus, the time required for generating **links** is $O(n)$, where n is the number of stems in the pattern. A simple search on links is performed to determine the position of ε which satisfies the above condition. This also requires linear time in the length of the pattern and consequently linear in the number of stems n. Thus, the total time required for this phase of the algorithm is $O(n)$.

3.1.2 Generating Grammar for a Single Pattern

The general idea of the grammar generation for a pattern is to choose the correct types of trees, from the TAG_{RNA} model, that can model dependencies between pairs of substring symbols in the pattern, or simply model independent substrings. The choice is dependent on the relative positions of the substrings being modeled and the position of ε. If we look at the types of trees in TAG_{RNA}, illustrated in Fig. 2, we notice the following. First, there is only one type of initial tree which is of TYPE 1. Thus the generated grammar will always have one of those trees. TYPE 2 trees can be used to model dependent pairs of substrings $(x, \overline{x^r})$ that appear on opposite sides of ε. TYPE 3 trees can be used to model dependent pairs of substrings $(x, \overline{x^r})$ that appear on the same side of ε. Finally, TYPE 4 trees can be used to model independent substrings (loops in the RNA structure) that are represented by w symbols in the generated pattern. As we mentioned above, we currently support patterns that have exactly one ε symbol. TYPE 5 trees can be used to model more complex structures with branching. At the moment, we do not make use of TYPE 5 Trees.

To generate the grammar for one pattern, the pattern is parsed one symbol at a time. For each independent substring symbol w or dependent pair of symbols $(x, \overline{x^r})$, two auxiliary trees are generated. The first tree has the same non-terminal label for the root, foot node and the adjoining node. This tree can be used recursively to generate terminals $\{c, g, u, a\}$ in the RNA sequence corresponding to the currently parsed pattern

substring symbol(s). The second tree is the same as the first one except that it has a different non-terminal label for the adjoining node. This tree allows transitioning to another substring or pair of substrings in the pattern. Generating a grammar for a single pattern requires linear time in the length of the pattern and, consequently, the number of stems is O(n). Algorithmic details and complexity analysis can be found in [1].

3.1.3 Final Grammar Composition

When the whole sample is processed, we will have a set of grammars, each representing the pattern of a single input example. To generate one grammar, which is representative of the whole input sample, we need to combine these grammars. The TAG union operator, defined in [27] can be used for this purpose. The union of two TAGs consists of the union of the elementary trees of both grammars.

If the input sample includes at least one sequence representing each RNA structure in the population being modeled, then the output grammar is a correct model for that population. For an input sample of size m RNA examples, the total time required by the algorithm is O(mn) where n is the maximum number of stems in an RNA example. Thus the algorithm is linear in the size of the input.

In order to reduce the size of the final grammar, the grammar composition step can be adjusted to check for input examples that have the same pattern. To accomplish that, any generated pattern must be saved. When a new example is encountered, a pattern is generated for it. Then, the set of saved patterns is searched. If the same pattern was generated before, we move to the next input example. If not, a grammar is inferred for the new pattern. The search process requires O(mn) time for one pattern. Thus, this modification increases the complexity to O(m^2n). Even though this is more than linear time, this algorithm is practical.

In practice, however, we prefer to keep the generated grammars separate. In later stages of the training phase and in the identification phase, the TAG parser will parse the input sequence against each of the generated grammars separately which is equivalent to parsing it against the union grammar. This will not increase the parsing complexity. On the contrary, it will help in optimizing it through eliminating the least effective grammars, as explained in section 3.3.

An Example

The input in this example is the following set of 4-tuples representing stems' positions for the delta ribozyme structure of the hepatitis delta virus (Italy variant) as it appears in the Pseudobase website [4].

(1,7,33,39), (16,19,81,84), (20,22,30,32), (43,49,68,74), (54,57,62,65)

First the corresponding pattern is generated:

$$x_1 w_1 x_2 x_3 w_2 \overline{x_3^r}\,\overline{x_1^r}\, \varepsilon w_3 x_4 w_4 x_5 w_5 \overline{x_5^r} w_6 \overline{x_4^r} w_7 \overline{x_2^r} w_8$$

Table 1 shows the output trees generated for each substring or pair of substrings in the above pattern. The substrings appear in the order in which they are processed by the algorithm.

Table 1. Output Trees for delta ribozyme structure of the hepatitas delta virus

Substring/Substring Pair	Generated Auxiliary Trees
$(x_1, \overline{x_1^r})$	T3L[S,S] & T3L[S,A]
w_1	T4Ld[A,A] & T4Ld[A,B]
w_8	T4Rd[B,B] & T4Rd[B,C]
$(x_2, \overline{x_2^r})$	T2d[C,C] & T2d[C,D]
$(x_3, \overline{x_3^r})$	T3L[D,D] & T3L[D,E]
w_2	T4Ld[E,E] & T4Ld[E,F]
w_3	T4Ru[F,F] & T4Ld[F,G]
w_7	T4Rd[G,G] & T4Rd[G,H]
$(x_4, \overline{x_4^r})$	T3R[H,H] & T3R[H,I]
w_4	T4Ru[I,I] & T4Ru[I,J]
w_6	T4Rd[J,J] & T4Rd[J,K]
$(x_5, \overline{x_5^r})$	T3R[K,K] & T3R[K,L]
w_5	T4Ru[L,L] & T4Ru[L,M]

3.2 The TAG Parser and the Scoring Function

We use a TAG parser in the training phase and the identification phase. In the training phase, the parser is used to generate a set of scores for the positive and negative training sequences. The generated scores are then input to a threshold function inference module. The scoring function used is a simple one that counts the number of base pairs for the sequence structure under a certain grammar. If there is more than one possible structure, due to the nondeterministic nature of the grammar, the parser will output the maximum score. As mentioned in section 3.2.3, a separate grammar for each pattern resulting from the positive training will be generated. The score for a certain sequence under the union of a set of grammars will, again, be the maximum of the scores generated from all grammars in the set.

The parser we used is an implementation of Rajasekaran's [17] and Vijay-Shankar and Joshi's [27] algorithms with some minor modifications. In our implementation of the TAG parser, in addition to n^4 matrix, A, maintained by the parser, we associate a list of 4-tuples with every node in the grammar. For a node α, a tuple $(i,j,k,l) \in List(\alpha)$ iff $\alpha \in A(i,j,k,l)$. This idea, borrowed from [17], does not improve the worst time complexity of the parser which is $O(n^6)$; however, it improves the average run time in practice due to sparsity of the matrix A. Another modification is the fact that the parser generates a score for each sequence instead of a yes/no output.

3.3 The Threshold Function Inference Module

This module infers a score threshold function $Th(l) = p$. A sequence s of size l is considered to have the RNA structure represented by a grammar G iff the TAG parser accepts s under G, with score $p_s \geq p$. Th(l) is a step function defined as follows:

$$Th(l) = p, \ i \leq l < j \tag{1}$$

Since both sensitivity and specificity are important criteria we infer a function Th(l) that maximizes the sum of sensitivity and specificity. This is achieved through calculating a function S for all possible paths of Th from l = 0 to l = n, where S is the maximum gain in specificity - loss in sensitivity resulting from each step the function Th makes and n is the maximum sequence size. Then Th is constructed by tracing back the path resulting in maximum S. Calculating maximum S can be done in $O(n^3m^2)$ time and $O(n^2m^2)$ memory using dynamic programming, where m is maximum reported score for the input sample.

Assume, with out loss of generality, that the number of sequences in the positive sample and the negative sample are equal. Let S(i,j,p,q) be maximum gain in specificity – loss in sensitivity possible for a threshold function segment that starts at Th(i) = p, and ends at Th(j) = q. Then, the dynamic programming recurrence formulae are given below

$$S(i,i,p,q) = S(i,i,q,q) = (\text{ the number of negative samples of length i with score} < q - \text{the number of positive samples of length i with score} < q) / \text{the sample size.} \quad (2)$$

and

$$\begin{aligned} S(i,j,p,q) &= S(i,i,p,p) + S(j,j,q,q) && , j = i+1 \\ &= \text{Max}_{\,p \le l \le m \le q} \left(S(i+1,j-1,l,m) + S(i,i,p,p) + S(j,j,q,q) \right) && , j \ge i+2 \end{aligned} \quad (3)$$

3.4 Selecting the Best Grammar Combination

As mentioned earlier, the scores resulting from each grammar for the patterns generated by the training sequences are reported separately. Instead of inferring the threshold function from the maximum score calculated over all the generated grammars, we try all possible combinations out of these grammars and pick the combination that generates the maximum sensitivity + specificity for the training set. This approach has two advantages. First, it eliminates the least informative and/or nearly redundant grammars. Meanwhile it enhances the time performance for the identification phase by reducing the number of grammars, or in other words, the size of the overall grammar.

This idea can further be used to restrict the number of grammars used to preset a maximum; thus choosing the best combination out of three or four grammars, for example. Even though trying out all possible combinations requires exponential time in the number of grammars, the number of grammars is usually small, resulting in the feasibility of this solution.

4 Experimental Results

To evaluate the effectiveness of the inferred grammars within $TAG_{RNA}Inf$, we calculate the sensitivity and specificity of identification.

$$Sensitivity = \frac{TP}{TP + FN} \text{ and } Specificity = \frac{TN}{TN + FP} \quad (4)$$

where TP, TN, FP, and FN are the number of true positives, the number of true negatives, the number of false positives and the number of false negatives respectively.

We used the grammar inference algorithm to infer a grammar for H-type pseudoknot from a positive training set with structural information. Then we used positive and negative training sets to infer the threshold function. The inferred grammar and score threshold function were applied to a test set of RNA sequences and the sensitivity and specificity were calculated.

For this experiment, we used these data sources:

- The positive data population of H-type pseudoknot sequences was collected from Pseudobase [4], the tmRNA database [28], and pseudoknot familes in the Rfam database [10]. We arbitrarily selected sequences from tmRNA and extracted PK1, PK2, and PK4 from them.
- The negative data population was driven from the Rfam database [10]. We selected non-pseudoknot families taking into consideration that the lengths of these sequences would be in the same range as the positive population.

The size of each population was 500 sequences. We randomly divided each of the data populations to three equal subsets: Training set, test set 1 and test set 2. Table 2 lists the sensitivity and specificity for each subset and for the whole population. Table 3 lists the sensitivity and specificity of TAG_{RNA}INF, TAG_{RNA} [26] and PknotsRG (mfe) [18] when applied to Test set 1. For TAG_{RNA}, and PknotsRG (mfe), we count TP to be the number of sequences belonging to the positive population with predicted structures exhibiting a pseudoknot. On the other hand, TN is the number of sequences belonging to the negative population with predicted structures not exhibiting a pseudoknot.

Results in table 2 indicate that our approach is solid and can result in very accurate predictions. The same problem has been addressed in [14] using a different grammatical formalism. However, the sensitivity we achieve is superior to that reported in [14]. For instance when the size of the training set is 50% of the available sample, they can achieve a sensitivity of only 54%. To achieve a sensitivity of 85%, they have to employ a training set of size 90% of the sample. They do not report specificity results. Results in table 3 indicate that our approach achieve a good balance between sensitivity and specificity.

Table 2. Experimental Results for TAG_{RNA}INF

Data Subset	Sensitivity	Specificity
Training set	87.4%	84.4%
Test set 1	78.4%	80.8%
Test set 2	79.6%	88%
Whole Population	81.8%	84.4%

Table 3. Comparative Results for Test set 1

	Sensitivity	Specificity
TAG_{RNA}INF	78.4%	80.8%
TAG_{RNA}	100%	71.3%
PknotsRG (mfe)	41.6%	81.4%

5 Conclusion

In this paper we have presented a grammatical inference algorithm for $TAG_{RNA,}$. We used the inference algorithm as a module within a complete RNA structure identification framework, TAG_{RNA}Inf, capable of identifying pseudoknot structures. The TAG parser used within TAG_{RNA}Inf utilizes a scoring function along with the inferred grammar. The scoring function currently used is the number of base pairs of the RNA structure detected by the parser. For a training set and a test set of equal size, our experimental results outperforms those reported in [14] for the same problem. They use a different grammatical model.

References

1. Al Seesi, S.: Pseudoknot Identification through Learning TAG_{RNA}, BECAT-CSE Technical Report, University of Connecticut (April 2008)
2. Akutsu, T.: Dynamic Programming Algorithms for RNA Secondary Structure Prediction with Pseudoknots. Discrete Applied Mathematics 104, 45–62 (2000)
3. Ambros, V., Bartel, B., Bartel, D.P., Burge, C.B., Carrington, J.C., Chen, X., Dreyfuss, G., Eddy, S.R., Griffiths-Jones, S., Marshall, M., Matzke, M., Ruvkun, G., Tuschl, T.: A Uniform System for microRNA Annotation. RNA 9(3), 277–279 (2003)
4. van Batenburg, F.H.D., Gultyaev, A.P., Pleij, C.W.A., Ng, J., Oliehoek, J.: Pseudobase: a Database with RNA Pseudoknots. Nucl. Acids Res. 28(1), 201–204 (2000)
5. Brazma, A., Jonassen, I., Vilo, J., Ukkonen, E.: Pattern Discovery in Biosequences. In: Honavar, V., Slutzki, G. (eds.) ICGI 1998. LNCS (LNAI), vol. 1433, pp. 255–270. Springer, Heidelberg (1998)
6. Buratti, E., Dhir, A., Lewandowska, M.A., Baralle, F.E.: RNA Structure is a Key Regulatory Element in Pathological ATM and CFTR Pseudoexon Inclusion Events. Nucl. Acids Res. 35(13), 4369–4383 (2007)
7. Cai, L., Malmberg, R., Wu, Y.: Stochastic Modeling of RNA Pseudoknotted Structures: a Grammatical Approach. Bioinformatics 19(supp. 1), 66–73 (2003)
8. Dirks, R.M., Pierce, N.A.: A Partition Function Algorithm for Nucleic Acid Secondary Structure Including Pseudoknots. J. Comput. Chem. 24(13), 1664–1677 (2003)
9. Gilbert, W.: The RNA World. Nature 319, 618 (1986)
10. Griffiths-Jones, S., Moxon, S., Marshall, M., Khanna, A., Eddy, S.R., Bateman, A.: Rfam: Annotating Non-coding RNAs in Complete Genomes. Nucl. Acids Res. 33, D121–D124 (2005)
11. Holbrook, S.R.: RNA Structure: the Long and the Short of it. Current Opinion in Structural Biology 15, 302–308 (2005)
12. Joshi, A.K., Levy, L., Takahashi, M.: Tree Adjunct Grammars. Journal of Computer and System Sciences 10, 136–163 (1975)
13. Laxminarayana, J.A., Nagaraja, G., Balaji, P.V.: Identification of Pseudoknots in RNA Secondary Structures: A Grammatical Inference Approach. In: Mukherjee, D.P., Pal, S. (eds.) Proceedings of 5th International Conference on Advances in Pattern Recognition (2003)
14. Laxminarayana, J.A., Nagaraja, G., Balaji, P.V.: Inference of a Subclass of Even Linear Languages and its Application to Pseudoknot Identification. In: Department of Computer Science and Engineering, Indian Institute of Technology, Bombay, India (manuscript, 2003)

15. Paillart, J.C., Skripkin, E., Ehresmann, B., Ehresmann, C., Marquet, R.: In vitro Evidence for a Long Range Pseudoknot in the 5'-Untranslated and Matrix Coding regions of HIV-1 Genomic RNA. J. Biol. Chem. 277, 5995–6004 (2002)
16. Pedersen, J.S., Bejerano, G., Siepel, A., Rosenbloom, K., Lindblad-Toh, K., Lander, E.S., Kent, J., Miller, W., Haussler, D.: Identification and Classification of Conserved RNA Secondary Structures in the Human Genome. Public Library of Science. Computational Biology 2(4), 33 (2006)
17. Rajasekaran, S.: Tree-Adjoining Language Parsing in $o(n^6)$ Time. SIAM Journal on Computing 25(4), 862–873 (1996)
18. Reeder, J., Giegerich, R.: Design, Implementation and Evaluation of a Practical Pseudoknot Folding Algorithm Based on Thermodynamics. BMC Bioinformatics 5, 104 (2004)
19. Rivas, E., Eddy, S.: The Language of RNA: a Formal Grammar that Includes Pseudoknots. Bioinformatics 16(4), 334–340 (2000)
20. Robertson, M.P., Igel, H., Baertsch, R., Haussler, D., Ares Jr., M., Scott, W.G.: The Structure of a Rigorously Conserved RNA Element within the SARS Virus Genome. Public Library of Science: Biology 3(1), 5 (2004)
21. Sakakibara, Y., Brown, M., Hughey, R., Mian, I.S., Sjolander, K., Underwood, R.C., Haussler, D.: Stochastic Context-Free Grammars for tRNA Modeling. Nucl. Acids Res. 22, 5112–5120 (1994)
22. Sakakibara, Y.: Grammatical Inference in Bioinformatics. IEEE Transactions on Pattern Analysis and Machine Intelligence 27, 1051–1062 (2005)
23. Searls, D.: The Linguistics of DNA. Am. Scient. 80, 579–591 (1992)
24. Takakura, T., Asakawa, H., Seki, S., Kobayashi, S.: Efficient Tree Grammar Modeling of RNA Secondary Structures from Alignment Data. In: Proceedings of posters of RECOMB 2005, pp. 339–340 (2005)
25. Tanaka, Y., Hori, T., Tagaya, M., Sakamoto, T., Kurihara, Y., Katahira, M., Uesugi, S.: Imino Proton NMR Analysis of HDV Ribozymes: Nested Double Pseudoknot Structure and Mg2+ Ion-Binding Site Close to the Catalytic Core in Solution. Nucl. Acids Res. 30, 766–774 (2002)
26. Uemura, Y., Hasegawa, A., Kobayashi, S., Yokomori, T.: Tree Adjoining Grammars for RNA Structure Prediction. Theoretical Computer Science 210(2), 277–303 (1999)
27. Vijay-Shanker, K., Joshi, A.K.: Some Computational Properties of Tree Adjoining Grammars. In: 23 rd Meeting of the Association for Computational Linguistics, pp. 82–93 (1985)
28. Williams, K.P., Bartel, D.P.: The tmRNA Website. Nucl. Acids Res. 26(1), 163–165 (1998)
29. Williams, K.P.: The tmRNA Website: Invasion by an Intron. Nucl. Acids Res. 30(1), 179–182 (2002)