# **CSCI 8610: Topics in Theoretical Computer Science in Computational Linguistics of Biomolecules**

# **Project 3**

# **Develop an optimal parser for stochastic k-tree grammar (SkTG)**

**Group Members:**

Arash Khoshparvar

Sina Solaimanpour

Seyed Navid Hashemi Tonekaboni

# **Introduction**

Stochastic context-free grammars are currently successful in modeling bio-molecular structures especially RNA secondary structure. But there is more than that to RNA structures and that is pseudo-knots and the RNA tertiary structure. It is known as a fact that a context-free grammar is unable to predict any type of pseudo-knots or tertiary structures. It also yields that the problem in general, needs a context-sensitive grammar which is an NP-hard problem. There has been some attempts to achieve some grammars that are able to predict at least a portion of the RNA tertiary structures. This is done by trying to eliminate extra branches of the original problem.

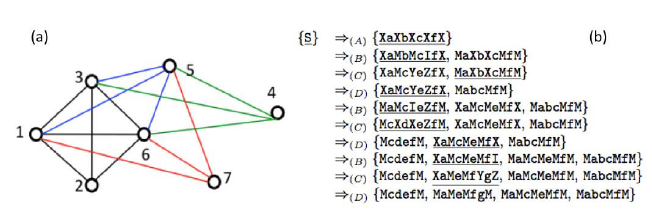
A Stochastic k-tree grammar is a type of grammar that builds strings with respect to relationships that are found in k-trees. Some of the existing linguistic grammars, developed mostly for natural language processing, are insufficient to account for crossing relationships incurred by distant interactions of bio-residues, while others are overly powerful and cause excessive computational complexity. A stochastic grammar, called stochastic k-tree grammar (SkTG), for the analysis of context-sensitive languages is introduced. With the new grammar rules, co-occurrences of distant terminals are characterized and recursively organized into *k*-tree graphs. The new grammar offers a viable approach to model context-sensitive interactions between bio-residues because such relationships are often constrained by *k*-trees, for small values of *k*, as demonstrated by earlier investigations.

# **Problem Statement**

We are going to develop an optimal parser for stochastic k-tree grammar (SkTG).  Let’s first define k-trees:

1. A k-tree of k + 1 vertices is a clique of k + 1 vertices
2. A k-tree of n vertices, for n > k + 1, is a graph consisting of a k-tree G of n − 1 vertices and a vertex v, which does not occur in G, such that v forms a new (k + 1)-clique with some size-k clique already in G.

The following figure shows of a 3-tree with seven vertices. The order in which 4-cliques formed is initially {1, 2, 3, 6}, vertex 5 added, then vertex 7 added, and finally vertex 4.



The stochastic k-tree grammar is based on definition of m-alternating strings. We call a symbolic string an m-alternating string, if it has the format X0a1X1 · · · amXm for some m ≥ 0, such that Xi ∈ N ∪ {ε} for all 0≤ i ≤ m and ai ∈ Σ for all 1 ≤ i ≤ m.

# **Method Design**

The algorithm to parse a stochastic k-tree grammar is a dynamic programming algorithm. We define a function F, with inputs that are: an m-alternating string and a set of indexes which show the places the terminals are going to show up in the input string which has to be parsed. This is a recursive algorithm but to avoid exhaustive search whenever any F is computed, it is going to be stored to avoid recalculations of the same F.

Let α = X0a1X1 …ak+1Xk+1 ∈ (Σ∪N)+ be a symbolic string and K = (l1, l2,…, lk+1) be k +1 ordered integers where 1 ≤ l1 < l2,…, lk+1 ≤ n. (α, κ) is a consistent pair if:

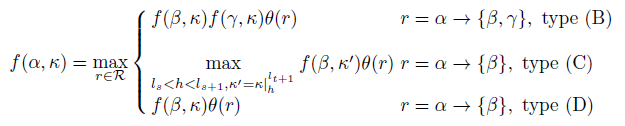
1. ai = sli, 1 ≤ i ≤ k + 1, and
2. For i = 0, 1,..., k + 1, Xi = ε iff li = li+1 − 1 (l0 =df 1 and lk+2 =df n).

Now given a pair (α, κ), we define function f (α, κ) to be the max probability for a derivation {α} ⇒∗T, where T ⊆ (Σ ∪ {M})+ for which *s* is the underlying string. Then function *f* can be recursively defined according to types of α, and the types of rules α is involved with in R.

1. α ∈ (Σ ∪ {M})+:

f(α, κ) =1 if (α, κ) is a consistent pair, otherwise it is 0

1. α ∈ (Σ ∪N)+ but α≠S:



where for the case of r being a type (C) rule, s and t are known values given in

satisfying (s-t) > 1 or (t-s) ≥ 1, and k' = k|lt+1 h  represents the ordered set modified from k by replacing lt+1 with h.

1. α = S:

f(S, k) = f(β, κ)θ(S → {β})

# **Implementation of Algorithm**

We have designed a dynamic programming algorithm in order to optimally parse the SkTG. We have also used Java map to avoid any redundant memory use. So only for calculated instances of F, one memory unit will be occupied. Here is the implementation of our java map:



As you can see, it is a map from Strings to BigDecimals. The String is in the format:



Where k is the set of indexes and rhs1.toString is an m-alternating string. Hence, if F is computed for a pair, it will be used and not re-calculated.

Obviously, the recursive part is implemented the same as what discussed in Method Design section: k, the set of indexes is updated whenever a rule of type BC (we have combined rules of type B and C together) is used.

# **Test Results**

In this section we present results from two simple applications we wrote for our parser.

The first application is a parse for the same example in presented in the Problem Statement section of the paper, for parsing “abcdefg”.

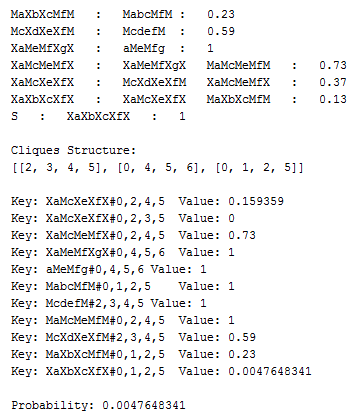


Figure 1 Results of parsing

The second one is a simple application for the RNA structure prediction.

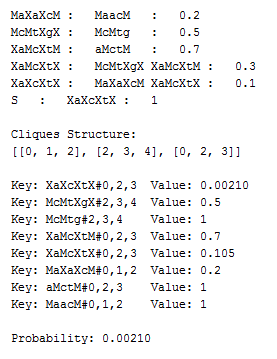


Figure 2 an application for RNA structure prediction

As you can see, the final probability and final cliques are both shown. The full-map is also printed to show memory use efficiency.

**Responsibilities**

Every member of the group had the equal amount of responsibilities in this project as we implemented the project together.