ALBERT-LUDWIGS-UNIVERSITÄT FREIBURG IM BREISGAU

Learning to design RNA polymers with graph kernels

by

Stefan Mautner

Master's Thesis

in the
Faculty of Engineering
Department of Computer Science

December 2015

Bearbeitungszeitraum

12. 06. 2015 – 12. 12. 2015

Gutachter

Prof. Dr. Rolf Backofen

Dr. Frank Hutter

Betreuer

Dr. Fabrizio Costa

Declaration of Authorship

Erklärung

Hiermit erkläre ich, dass ich diese Abschlussarbeit selbständig verfasst habe, keine anderen als die angegebenen Quellen/Hilfsmittel verwendet habe und alle Stellen, die wörtlich oder sinngem aus veröffentlichten Schriften entnommen wurden, als solche kenntlich gemacht habe. Darüber hinaus erkläre ich, dass diese Abschlussarbeit nicht, auch nicht auszugsweise, bereits für eine andere Prüfung angefertigt wurde.

Unterschrift: ______________________________

Datum: ______________________________

Zusammenfassung

RNA Polymere haben viele wichtige Funktionen in Zellen und sind einfach zu generieren. Darum ist es wünschenswert RNA Sequenzen zu konstruieren die bestimmte Funktionen erfüllen. Konventionell wird dafür auf *inverse folding* gesetzt. Inverse folding ist jedoch auf Benutzer gegebene, strenge Constraints angewiesen. Hier präsentiere ich eine Herangehensweise, die diese Constraints lernt bevor neue RNA Moleküle generiert werden.

F. Costa [1] beschreibt einen universellen Ansatz um Proben aus Verteilungen über Graphen zu entnehmen. Dieser benutzt Metropolis Hastings sampling zusammen mit einer Graphen Grammatik um Graphen zu generieren und einem Graphen Kernel um die generierten Graphen zu bewerten.

Da die beschriebene Grammatik lokal sensitiv ist, können in ihr keine weitreichenden Abhängigkeiten abgebildet werden. Dies könnte zu einer erhöhten Zurückweisungsrate des MH Algorithmus führen.

Dies gilt besonders für RNA Moleküle, da hier durch die Sekundärstruktur bedingt, weitreichende Abhängigkeiten bestehen können. In dieser Arbeit präsentiere ich eine Methode die Graph Minors (ein Graph Minor entsteht durch zusammenziehen von Kanten in einem Graph) benutzt um die Grammatik zu erweitern und ihre Ausdrucksfähigkeit zu steigern.

Abstract

RNA polymers have many functions in cells and are relatively easy to create. Finding ways to construct RNA sequences that correspond to a desired function is thus desirable. Conventionally *inverse folding* is used to tackle this problem. Inverse folding however is dependent on user provided very strict constraints. In this work I present an approach that will *learn* those constraints before generating new RNA molecules.

[1] describes a general purpose approach to sample distributions of graphs. Metropolis Hastings sampling is used with a graph grammar to propose graphs and a graph kernel to aid the evaluation of proposed graphs.

Since the described grammar is locally context-sensitive, long range constraints can not be represented in the grammar. If only local constraints are considered when proposing a graph, ignoring long range dependencies will increase the rejection rate of the MH algorithm.

This is especially true for RNA secondary structure graphs. The structures themselves impose constraints on the graphs that cant be represented in a locally context-sensitive grammar. In this work I present a method which is using graph minors (a minor is obtained by contracting adjacent nodes in a graph) to enhance the grammars expressiveness.

Contents

Chapter 1

Introduction

Ribonucleic acid (RNA) molecules have many functions in cells of living organisms, they are created by transcribing DNA. RNA sequences can be translated into proteins or fold into a structure that determines its effects.

Sequences whose Structures exhibit similar effects can be grouped into classes. In this work I will present a method that takes one of these classes and tries to generate RNA sequences whose structure will behave similar to the sequences of the input class.

The conventional method of calculating an RNA sequence for a desired function is *inverse folding*. With inverse folding one assumes a structure tries to find the sequence(s) of nucleotides that yields the minimum free energy(MFE)[2], i.e. is most stable, in this structure.

Many Algorithms have been described that try to solve this problem. For complexity reasons most algorithms, e.g. INFO–RNA [3], are not able to deal with pseudo knots and only consider the secondary structures, instead of tertiary (3D) structures of RNA.

1.1 Advantages over inverse folding

Inverse folding exhibits additional problems that the approach presented here is not prone to. Rather than declaring a secondary structure for inverse folding, we suggest to input examples so that the constraints is learned before constructing RNA sequences.

Inverse folding assumes that we know the secondary structure constraints that depict the function of an RNA sequence exactly. Some of these constraints might not be known yet or biologically wrong. By learning the constraints we are avoiding these issues.

In inverse folding one starts out with a very restrictive way of creating new RNA sequences. This makes it hard to implement new constraints quickly. My approach to construction however is a generic one. Building a filter is in general easier than extending what is allowed to be built.

In addition many parts of my algorithm can be tempered with easily. I will give some examples of ways to do this. In the normal version of the algorithm, the goal structure and sequence are leraned. One might replace this goal with a similarity meassure to a custom secondary structure graph. The generation stage of the algorithm will make use of fragments of secondary structure graphs to alter parts of a graph. These fragments are collected in the initialization stage and can be changed freely. One might also be interested in sequences with limited size, in this case it is easy to control the size of the whole graph by filtering the used fragments by size.

1.2 Sampling a distribution of graphs

F. Costa [1] describes a way to sample graphs that belong to a distribution over graphs. One provides a set of graphs as input on which we train a one class machine learning method to estimate the distribution of the input graphs. This means that the trained machine learning model will assign good scores to graphs that are similar to the input and worse scores to any graph that is not related to the input.

When it comes to sampling, the seed instances are altered with subgraphs that were collected in the initialization stage. With the model the quality of our created instance is estimated. An instance with lower probability than its predecessor is only accepted with a certain probability. This is can be viewed a variant of the Metropolis Hastings (MH) algorithm.

This is one of the first approaches to learn how to sample graphs that have certain properties. As it is a generic approach, it can be used on many types of structured data.

1.3 Contribution

One of these structured data types may be the secondary structure of RNA sequences. The algorithm described by Costa [1] however, does not cope well with long range dependencies in graphs since only small, very local, subgraph fragments are considered when altering a graph instance. The metropolis hastings algorithm would spend most of its time rejecting instances that have a low probability. Working with graph minors (resulting from edge-contractions of a graph), I devised a method to overcome this limitation. Long range dependencies as observed in secondary structure of RNA can now be depicted.

1.4 The structure of RNA

I will give a short introduction to RNA. RNA sequences are sequence of nucleotides chained together, with one end being the distinct start. Nucleotides are adenine (A), guanine (G), cytosine (C) and uracil (U). Nucleotides are not only linked to their neighbor in the sequence, but may also form a hydrogen bond with another nucleotide. Allowed are the base pairs G-C and A-U. Base pairings from certain structure elements when viewed on the level of the whole sequence. The elements that are relevant in this work are shown in figure 1.1.

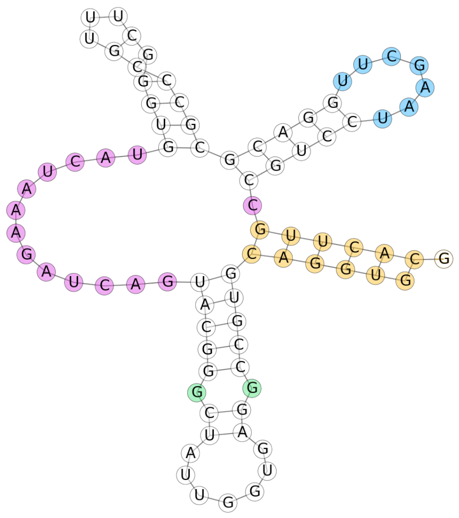

FIGURE 1.1: Overview of structural elements of an RNA secondary structure. we can see a *hairpin loop* (blue), an *internal loop* (green), a *multi loop* (pink) and a *stem* (orange)

Chapter 2

Method

In this chapter I discuss how to learn to design RNA polymers. The algorithm takes instances of RNA polymers (e.g. an rfam-family [4]) in input. It will estimate a distribution over those instances and yield polymers that are close to the dense region.

2.1 Basic sampling

This thesis is based on work by F. Costa [1]. He describes how to sample graphs according to the distribution of a given set of graphs. We will later see why this approach is lacking when it comes to RNA secondary structure graphs and how we can improve it.

2.1.1 The learning problem

Costa [1] defines the success of sampling of a graph distribution in terms of an optimization problem: The *Constructive Learning Problem*, **CLP**. First we need a few definitions. G_0 is a set of graphs, the input for the algorithm. F_γ is a function to estimate a probability distribution. M_θ: M is a function that maps a set of graphs to another set of graphs, its parameters are θ. L is a loss function between distributions (equal distribution should yield maximum loss score).

$$\hat{\theta} = \arg\max_{\theta} \ L(F_\gamma(G_0), F_\gamma(M_\theta(G_0)))$$

Intuitively the **probability distributions over graphs** $F_\gamma(G_0)$ is a density function that assigns a high value to graphs that are similar to the set G_0 and a low value otherwise.

We try to minimize the difference between the input distribution and the generated distribution. The remaining problem is that if M_θ is the identity function, then its outputs will be an optimal solution. To penalize generated graphs that are similar to input graphs, another term is introduced – with λ for weighting and K as a kernel function to measure similarity between graphs.

$$\hat{\theta} = \arg\max_{\theta} L(F_\gamma(G_0), F_\gamma(M_\theta(G_0))) - \lambda \frac{1}{|G_0||M_\theta(G_0)|} \sum_{G \in G_0, G' \in M_\theta(G_0)} K(G, G')$$

2.1.2 The Metropolis Hastings algorithm

M_θ is a procedure that produces graphs. The **CLP** dictates that these graphs should be sampled according to a distribution. This means that Graphs that belong to dense regions are produced more often than graphs that do not. The Metropolis Hastings algorithm is a Markov chain Monte Carlo method that is able to sample according to a distribution [5]. We will show how we can apply the Metropolis Hastings scheme to graphs.

In a Markov Process there is a probability for each state to switch to any other $P(x \to x')$. A *stationary distribution* is the distribution over the states after an infinite amount of state changes. We want the stationary distribution to be similar to the desired distribution. In the case of graphs this is the distribution over G_0. A Markov process converges to a stationary distribution if it is *ergodic* and has *detailed balance*. Detailed balance is achieved when $P(x \to x') = P(x' \to x)$ for all states x and x'. To be ergodic the process must 1) always be able to reach any state and 2) not reach any state in fixed intervals.

In the MH Algorithm, we pick a state (in our case this will be a graph $\in G_0$) and transition with $P(x \to x') = g(x \to x')A(x \to x')$ to another state.

$g(x \to x')$ is the probability of proposing x' when in state x and $A(x \to x') = \min(1, \frac{P(x')}{P(x)} \frac{g(x' \to x)}{g(x \to x')})$. If $A(x \to x') > 1$, x' is accepted otherwise we accept the proposition with the probability $A(x \to x')$. According to Metropolis(1953) [5] the states we will visit this way belong to the desired distribution. In the case of graphs this will be $F_\gamma(G_0)$.

The Metropolis Hastings algorithm for graphs

Concretely, the acceptance function for graphs is $A(G \to G') = \min(1, \frac{P(G')}{P(G)} \frac{g(G' \to G)}{g(G \to G')})$. $P(G)$ is the likelihood of G to belong to the distribution G_0. $g(A \to B)$ is the probability of suggesting B when looking at graph A. Although going from B to A and back is always possible, $g(A \to B)$ is likely not equal $g(B \to A)$.

This is essential for the scheme as it impacts the acceptance function. In the case of normal graphs, one can estimate the error and adjust the acceptance function. The correction scheme however, is not applicable for RNA graphs and therefore my sampling process resembles simulates annealing[6].

2.1.3 Probability distribution over graphs

We estimate the probability distribution that will be used in the sampling of graphs via a *one class SVM*.

Elementary graph definitions

A graph G consists of two sets V and E. V is a set of *vertices*, sometimes referred to as *nodes*, while E consists of elements from $V \times V$ which are called *edges*. Vertices may have a label which is accessible via the label function *label*. A path is a series of consecutive edges that connect two nodes. Two graphs are *isomorphic*, $G_1 \cong G_2$ if there is a bijection ϕ between the sets of vertices of G_1 and G_2 such that for every edge (u, v) in G_1 there is an edge $(\phi(u), \phi(v))$ in G_2 and the label is preserved $label(u) = label(\phi(u))$ for all vertices u.

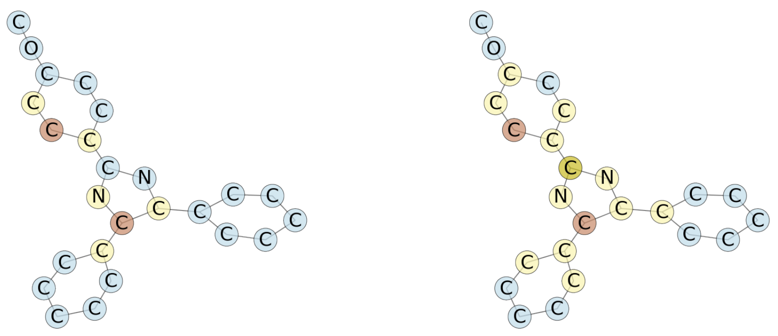

Figure 2.1: we observe two subgraph pairs that will each form a feature in NSPDK. We can observe root nodes in red that are in distance 4, and nodes in radius one (two) around them.

A graph kernel

A Support Vector Machine is a form of supervised learning which can not work with structured data types such as graphs. A Kernel is a function that works on graphs, since an SVM relies only on calculations between instances, kernel functions enable us to apply an SVM to graphs. We use the Neighborhood Subgraph Pairwise Distance Kernel (NSPDK) [7].

NSPDK will calculate a feature vector for every graph. This vector can then be compared to the vector of another graph. The kernel works by extracting subgraphs and using them as features. Subgraphs will be hashed so the number of features is limited by the bit-length of that hash. The number of subgraphs that will be extracted is limited by the number of nodes in a graph and a parameter that tells us how many subgraphs per node are gathered.

We will extract these subgraphs according to the following formula:
$R_{r,d}(G) = \{(N_r^u(G), N_r^v(G)) : d(u,v) = d\}$

$N_x^y(G)$ indicates the subgraph of a graph G that includes the *root* node y and all nodes around it in radius x; that are all notes that are reachable in a path of length x or less from the *root* node y.

$d(a,b)$ indicates the length of the shortest path between the nodes a and b in G. We extract all pairs of subgraphs (with radius r) whose root nodes are in the exact distance d of one another. In 2.1 we can observe two elements of $R_{r,d}(G)$.

If we want to compare two graphs, we count the number of subgraph-pairs they have in common. $\cong$ indicates isomorphism between subgraphs and "1" the indicator function. Thus the basic kernel is defined:

$$K_{r,d}(G,G') = \sum_{\substack{A,B \in R_{r,d}(G) \\ C,D \in R_{r,d}(G')}} 1_{C \cong A} \cdot 1_{D \cong B}$$

For r and d we may assume different values, so we calculate them up to a maximum:

$$K_{r_{max},d_{max}}(G,G') = \sum_{r=0}^{r_{max}} \sum_{d=0}^{d_{max}} K_{r,d}(G,G')$$

Costa [1] also describes normalization step. Since we are only counting the number of subgraph pairs that two graphs have in common larger graphs will obviously score higher than smaller graphs, simply because they contain more possible subgraphs.

An example of this would be a graph G' that contains two copies of a graph G. In that case $K_{r,d}(G,G) < K_{r,d}(G,G')$ will hold in all but the most trivial cases.

$$\hat{K}_{r,d}(G,G') = \frac{K_{r,d}(G,G')}{\sqrt{K_{r,d}(G,G)K_{r,d}(G',G')}}$$

$$\hat{K}_{r_{max},d_{max}}(G,G') = \frac{K_{r_{max},d_{max}}(G,G')}{\sqrt{K_{r_{max},d_{max}}(G,G)K_{r_{max},d_{max}}(G',G')}}$$

Subgraph encoding

For the feature extraction it is important that isomorphic subgraph pairs receive the same identifier. Each subgraph pair may be represented by the lexicographically ordered, hashed triplet of $< d, H(L^g(A)), H(L^g(B)) >$ with A and B as the subgraphs and d as the distance between the root nodes.

$L^g(G)$ will compute an identifier for subgraph G. It will later be hashed by a (Merkle-Damgård [8] type) hash function H. To compute $L^g(G)$, we first assign a new label $nl(v)$ to each node v. We do this by sorting the set $\{(d(v,u), label(u)) | u \in G\}$ lexicographically. For each edge (u,v) in G we can now list the triplets $< label((u,v), nl(u), nl(v)) >$ and sort them choreographically.

One class SVM

Here I am explaining how Costa [1] is using an SVM [9] to estimate a distribution over graphs. We already know how to vectorize graphs, what is missing is a way to estimate the quality of those vectors.

A Support vector machine is a form of supervised machine learning. It aims to find a hyperplane that separates the training data according to its classes.

A hyperplane in d dimensions can be described by the set of points $x \in \mathbb{R}^d$ that satisfy $w \cdot x - b = 0$, with $w \in \mathbb{R}^d$ being a normal vector to the hyperplane and $b \in \mathbb{R}$ being a constant.

We work with a training set of data points $\{(x_1, y_1), ...(x_m, y_m)\}$ with instances $x_i \in \mathbb{R}^d$ and associated class $y_i \in \{-1, 1\}$ for $i \in \{1, ..m\}$. We want to find a hyperplane that separates the data points such that for each point $y_i(w \cdot x_i - b) \geq 1$ holds. While doing so we want to maximize the distance of each point from the hyperplane *margin*. This can be achieved by minimizing $||w||$.

It turns out that finding a solution is not hard since it is in a convex space and can be found with gradient descend. Also finding the solution and using it to predict future instances depends only on the dot product of instances. This is why we can use kernel functions conveniently.

SVMs typically support *soft margins*, meaning outliers are allowed in the margin. In that case the constraints change to $y_i(w \cdot x_i - b) \geq 1 - \xi_i$, with ξ_i representing the amount of 'wrongness' in a training data-point in terms of distance from the plane. With that we out task becomes to calculate $\min\limits_{w \in \mathbb{R}^d, \xi \in \mathbb{R}^m, b \in \mathbb{R}} \frac{1}{2}||w||^2 + C \sum_i \xi_i$. the variable C determines the trade of between fewer outlines versus larger margin.

This was a short summary of a two class svm since all the training instances were either in class 1 or -1. Lets explore the **one class SVM**. If we are given a set of instances without a corresponding class, the task is to identify densities and punish instances that have few things in common with other instances.

In order to train a SVM we need a negative set to our one class $Z^+ = \{(x_1, 1), ...(x_m, 1)\}$. we create this by negating the first class: $Z^- = \{(-x_1, -1), ...(-x_m, -1)\}$. After training an SVM on Z^+ and Z^- we can change the position of the hyperplane so that it separates the original data. In 2.2 we can observe how the planes should behave.

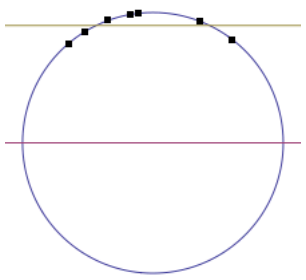
FIGURE 2.2: vectorized graphs (black squares) are normalized and thus lay on a hypersphere (blue). The red hyperplane was learned by mirroring the instances to the bottom of the sphere. The plane in yellow is calculated by moving the red hyperplane such that it 'cuts' the original data-points

If classifies an instance with a trained SVM it will calculate the distance to the plane. Platt [10] describes a way to transform this distance into the probability that the instance belongs to one of the classes.

2.1.4 Graph grammar

A formal grammar contains production rules that define its power. A Context sensitive grammar may include a production rule like $rAl \rightarrow rBl$. This should give an intuition for the context sensitive graph grammar that I am going to describe.

A core graph $C_R^v(G)$ of G is referring to a subgraph $N_r^v(G)$ as defined earlier with *radius* R starting in root v. An interface graph $I_{R,T}^v(G)$ is the difference between $C_{R+T}^v(G)$ and $C_R^v(G)$. We call T *thickness*.

$I_{R,T}^v(G)$ and $C_R^v(G)$ build a core interface pair (CIP) for G. CIPs G and G' (with $I_{R,T}^{v'}(G')$ and $C_R^{v'}(G')$) are *congruent* if their interfaces graphs are isomorphic (with isomorphism function ϕ) and $\forall u \in I_{R,T}^v(G) : \min_{z \in C_R^v(G)} d(u,z) = \min_{z' \in C_R^{v'}(G')} d(\phi(u), z')$ i.e. the distance to the closest core node is equal for every u and $\phi(u)$.

Isomorphism alone is not enough. Imagine an interface that consists of one edge (u,v) with $label(u) \neq label(v)$. With the formula above makes sure that if v is close to the core in G a congruent interfaces $\phi(v)$ is also close to the core and not on the far side. This is illustrated in figure 2.3.

[11] Describes a graph grammar as a finite set of productions $S = (M, D, E)$. A production consists of a *mother* graph M that will be replaced in a *host* graph

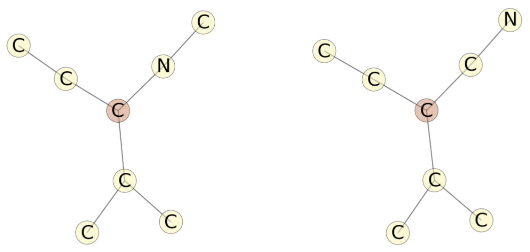

Figure 2.3: The subgraphs induced by the interface nodes (yellow) are isomorphic but not *congruent*.

H by the *daughter* graph D via the *embedding mechanism* E. A production is applicable if H contains a subgraph that is isomorphic to M. The subgraph M is then removed from H to obtain H^-. Then the daughter graph D is added to the graph H^- using the embedding mechanism E.

In this case M and D are congruent CIP graphs while E is given by the bijection between the Interface graphs of M and D. In other words a production is identifying a CIP in H and replacing it with a congruent CIP. All we need to do is finding the bijection between the interfaces of the CIPs that need to be swapped and we can rewire the graph.

We use the same hashing scheme described above again to create identifier for interface graphs of CIPs to make it easy to find applicable productions. If two CIPs exhibit the same identifier, they are congruent (except for hash collision) and thus the production is applicable.

In practice a grammar is created by extracting CIPs from an input set of graphs. After the grammar is obtained, the sampling can start. During sampling, productions are applied. In a production step only the core part of M is replaced. The new core has been seen in its new surrounding before because D is from G_0 This contributes to the quality of the proposed graphs to the Metropolis Hastings sampling algorithm.

A production is applied by first finding the isomorphism between the Interfaces of the CIPs M and D. Then the core nodes of M are deleted and for all interface nodes v, v is contracted with $\phi(v)$. We can observe this process in figure 2.4

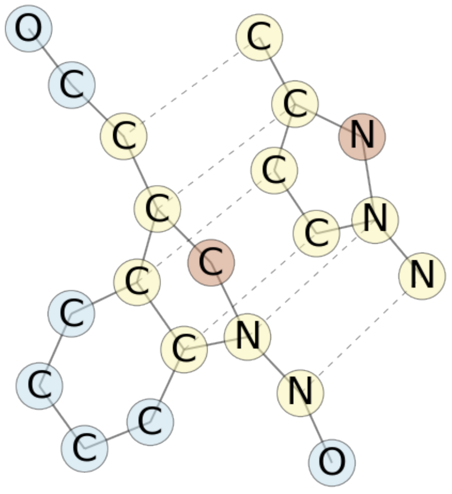

Figure 2.4: We see a host graph and a new CIP graph. The core (red carbon) of the mother M has not been removed yet. We see the isomorphism between the interfaces in dashed lines.

2.1.5 Algorithm summary

The algorithm has an initialisation phase, in which all input graphs become vectorized by the kernel. After this a one class SVM is trained on these instances whose hyperplane seperates the outliers from the dense region. Additionaly all possible CIPs are collected to create a grammar. All possible CIPs means a CIP for every node in every graph, vor every radius and every distance.

During sampling, input graphs (which might be the set we used for initialization) are subjected to a set number of substitutions. A substitution consists of choosing a CIP and applying a substitution with a congruent CIP from the grammar. After substitution we rate the credibility of the new graph via the one class SVM. we compare this score with the score of the graph we started out with and keep the new one with a certain probability. According to Metropolis this procedure will yield graphs that have similar probabilities than those in input.

2.2 Sampling with graph minors

A graph minor G' of a graph G is obtained by deleting edges or nodes and or by contracting edges. Although Costa and I only used edge contractions, other operations are possible.

Previously we have seen that the only constraint for a replacement of congruent CIPs is the interface graph. An interface graph is a very inflexible construct. Its reach is very limited. Greater thickness reduces the number of candidates for congruency drastically, especially since in the interface graphs every node label needs to match.

Imagine an interface on the level of graph minors. A node here might represent many nodes while its label might still be trivial. This way the grammar becomes more powerful, its reach increases. A method to do this is presented in this work.

2.2.1 Kernel support

A note in G' represents one or more nodes in G. During contraction we keep a set of contracted nodes, from G for each node in G'. we call these sets s_i for all vertices i in G'

It is possible to make a joint graph of G and G' with additional *contraction edges* between the node of s_i and all the nodes in G that it contains.

The implementation of the NSPDK that I use allows for the neighborhood graphs to ignore *contraction edges* while the connection between neighborhood graphs may cross them.

This means that the kernel can make use of graph minors which might contribute to the quality of the model learned by the SVM.

2.2.2 Building a grammar

In the basic grammar we picked a root node, a radius and a thickness to extract a CIP. A CIP is defined by the subgraph induced by the root and all nodes in distance *radius+thickness.*

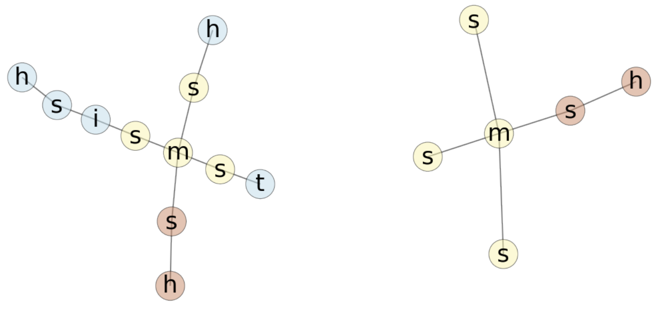

FIGURE 2.5: A minor graph and an extracted CIP created from t-RNA, (core:red, interface:yellow), the labels are abbreviations for the structural elements (fig. 1.1)

When working with not only a base graph G but also with a graph minor G' we pick a root node v in G', a radius R and thickness T for G' and a *base-thickness* B for G.

For G' we calculate the CIP as if it was a base graph, namely $C_R^v(G')$ and $I_{R,T}^v(G')$ (see figure 2.5). For G the core graph $C_R^v(G', G)$ is defined by the subgraph induced by the nodes $\bigcup\limits_{u \in C_R^v(G')} s_u$ of G, that are the nodes contracted by core nodes of G'. The base interface graph $I_{R,B}^v(G', G)$ is then defined by the nodes $\{w | d(w, v) \leq B \land v \in C_R^v(G', G) \land w \in G \land w \notin C_R^v(G', G)\}$ which are all the nodes in radius B around the base core nodes that are not base core nodes themselves.

The finished CIP graph is $I_{R,B}^v(G', G)$ and $C_R^v(G', G)$. For the embedding mechanism we use the same as before, since there is still an base interface-graph to be matched congruently.

We want to make sure that if we replace a CIP not only the base interface matches but also the interface of the minor graphs. To achieve this we use a hash function to combine the hash value of the graphs $I_{R,T}^v(G')$ and $I_{R,B}^v(G', G)$ together to obtain the interface identifier. Figures 2.6, 2.7 and 2.8 show examples of a CIP of G' and their partners in G.

Notice that for **applying** a production we work with the CIP graph ($I_{R,B}^v(G', G)$ and $C_R^v(G', G)$), but when we **look for** a production $I_{R,B}^v(G', G)$ and $I_{R,T}^v(G')$ need to be congruent. This is what increases the accuracy of the grammar.

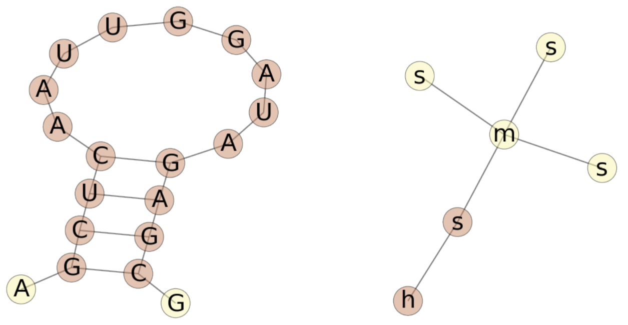

FIGURE 2.6: A hairpin (h) in G' was chosen as the root with $R = 1$, $B = 1$ and $T = 2$. We observe the cores in red and the interfaces in yellow.

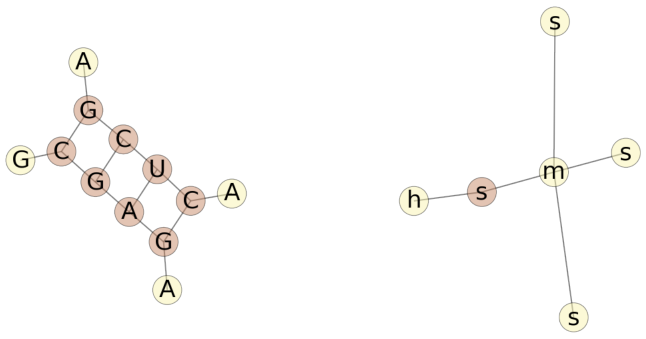

FIGURE 2.7: A stem (s) in G' was chosen as the root with $R = 0$, $B = 1$ and $T = 2$.

2.2.3 Generating graph minors

The method described here depends on graph minors. Graph minors can be acquired through a dedicated, domain specific algorithm or may even be learned.

A learned computation of graph minors is desirable to keep the sampling approach generic. I am going to talk about one way this could be done. [12] describes a way to associate nodes with weights. Contracting nodes according to their weights could yield a meaningful graph minor.

Although a learned way of generating graph minors is desirable, I have not found one yet. Therefore I work with a graph minor that connects the secondary structure elements as seen in fig. 2.8.

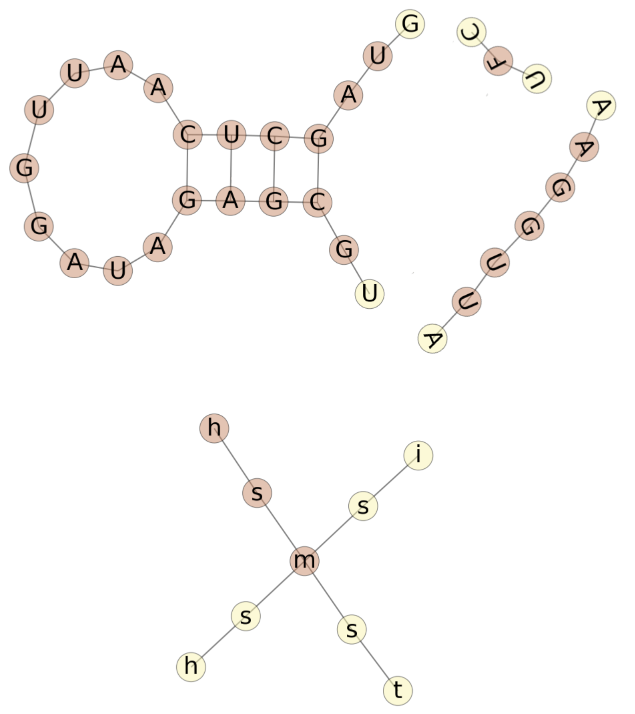

FIGURE 2.8: A stem (s) in G' was chosen as the root with $R = 1$, $B = 1$ and $T = 2$. This example is more complex, you can see that in the full graph stems would start at the A-U and G-C bonds. The F 'nucleotide' is a placeholder that might be substituted for a real multiloop part.

2.3 Constructing RNA

Finally I am going to discuss we can make the sampling process work for RNA.

2.3.1 Properties of RNA and minor creation

An RNA sequence has two distinct ends, the 5' and the 3' end. I keep this distinction in the graph by using a directed instead of an undirected graph during grammar operations. In a directed graph, edges are replaced by edges that have a distinct start and end vertex. Hydrogen bonds are depicted by both way directed edges. Since our kernel does not support directed graphs, we convert every graph before handing it to the kernel.

The minor graphs seen here are produced by contracting the graphs according to their secondary structure elements. The CIP in fig. 2.7 illustrates this nicely. Notice how in the minor graph we see a single node labeled 's' for stem, while in the original graph we see the all the stem vertices that had to be contracted to obtain it.

Multi-loops (as seen in fig. 2.8) and internal loops present a challenge if one chooses to employ the minor creation method I use. For all other structures one can identify the associated nodes and contract them to obtain the minor node for that structure.

As seen in the figure, the core nodes in G are not connected. To preserve the order these non connected *shards* are numbered so substitutions will preserve this order. Another interesting case is, two stems leaving the multi loop without a node in between. In this case I choose to add an empty nucleotide, that I named 'F'. With this we know on the level of G how many stems are connected to a multi loop. This case can be observed in figure 2.8.

2.3.2 The sampling process

RNA sequences fold into structures. The first operation applied to a sequence when we start to work with it is turning it into a secondary structure graph. We need to be aware that a change in the sequence may change the secondary structure graph. Therefore we recalculate the secondary structure after each sampling step.

We established earlier that the Metropolis Hastings algorithm requires *detailed balance* between two states, i.e. $P(x \rightarrow x') = P(x' \rightarrow x)$. Due to the refolding it might happen that the way back is not possible anymore. Imagine I substituted a CIP A with a CIP B. The core of CIP B is a stem. After refolding this stem might not exist anymore. Therefore sampling a distribution is made impossible by applying refolding. Since I am applying an increasing penalty for worse instances during sampling, my process resembles simulated annealing [6] in contrast to the original process described by Costa [1].

Refolding is not an easy task. Calculating the most stable [2] structure for a sequence does often yield a wrong structure. During initialization I prepare a nearest neighbor search on the input sequences. When it comes to folding, a sequence is folded together with its four closest known neighbors, resulting in a reliable secondary structure.

Nearest Neighbor: Finding the nearest neighbors is done by applying the the NSPDK [1] to a path graph whose labels correspond to the nucleotides of the sequence. Finding the nearest neighbor is done via brute force since the vectors are high dimensional and there is (to my knowledge) no optimized solution known.

Folding: The folding task is to find a consensus structure for the sequences i.e. a structure that every sequence can adhere to while minimizing edit operations on the sequences. We do this by first finding a *multiple sequence alignment*(MSA) with muscle [13]. Finding a sequence alignment means finding minimum cost set of edit operations (delete, insert, alter) such that one can change a sequence into another. Calculating a MSA means doing this for multiple sequences at once. After the MSA is determined, we need to calculate a structure that the aligned sequences will fold into. This *consensus structure* can be calculated using RNAalifold[14]. Note that this approach to folding might work less good if the input sequences are too long.

After folding we are left with a dot bracket string. From a dot bracket string, we can see for each nucleotide in a sequence if it is connected to another nucleotide via a hydrogen bond. A hydrogen bond is depicted as a opening and a related closing bracket. We can easily create a graph from this structure. First create a path graph with the same length as the sequence, then label it according to the sequence. Connect nodes whose nucleotides are bound together by hydrogen.

2.3.3 Summary of modifications and runtime

We observe differences to the method presented by Costa.

CIPs are vastly different from the CIPs introduced before. With the help of graph minors, cores may assume any shape. Before core nodes were limited to a radius around a root. Due to the directed nature of RNA, my implementation is working with directed graphs throughout. With the additional information of interfaces in the minor graph, we are able to find more precise CIPs for productions that can respect long range structural dependencies.

Since we are working with RNA, we need to make sure that in each state, the assumed structure corresponds to the sequence induced by the labels of the backbone nodes. For this reason we introduced a refolding scheme. Not only does this prevent us from sampling according to Metropolis Hastings, but this also has a negative effect on the runtime. The calculation of the minor graph is done in every step after refolding and its complexity depends on the graph minor scheme used. Since it is only depending on nodes in the graph, it is unlikely to be as slow as the following steps. RNAalifold has a runtime of $O(Nn^3)$ with N being the number of sequences that are to be aligned and n referring to the length of the sequences. For finding those N sequences we used a Brute force approach which is searching through every feature of every sequence in input. One sequence exhibits $n^2 * d * r$ (n – sequence length; d – distance of subgraphs; r – radius) features.

Chapter 3

Evaluation

The Rfam [4] database groups RNA sequences into families by function. For each family, representative members, seeds, are listed which were used to train a *covariance model* [15]. A covariance model is based on probabilistic context free grammars (PCFG), which are in turn equivalent to an extended version of hidden Markov models. The model can be used to identify potentially new RNA sequences that belong to the family the model was trained on. For each potential finding we are given a log odds scores. It is a binary logarithm of how many random sequences are as probable as the observed one to be in the family the model was trained for.

Rfam is offering a threshold for the trustworthiness of log odds score. This threshold is listed for each family and has been determined by human experts. The idea is, that if my method is creating a sequence that will score above this threshold, it is an indicator that the method has done well.

To train my method, I use the RNA families RF00005 (t-RNA), RF00162 (SAM riboswitch) and RF01725(SAM 1/4 variant). I chose those because they have over 400 seed sequences in Rfam, exhibit a similar length and have a non trivial but similar structure.

3.1 Parameters

I used the same parameters for all experiments.

Initialisation: The one class SVM will place 33% of instances in the negative class. This should not have much impact on the sampling quality, but it will influence the average score during sampling. For radius_list R I picked 0 and 1. Since this is the radius of the core graph in the minor graph, larger values dont make sense because due to the size of my minor. Set the thickness_list T to two only which is large compared to the minor graph. I choose a large value to increase the impact of my addition to the sampling process. For the base_thickness_list B I chose 1 to limit its impact as much as possible.

sampling: there are many sampling options but I will still give a short overview over the most important ones.
n_samples=3, we extract up to 3 samples in regular intervals.
n_steps=50, we conduct 50 sampling steps.
quick_skip_orig_cip=True, there is usually more than one congruent CIP in the grammar, choose new CIP in the graph once one did not work out.
burnin=10, ignore the first 10 steps when calculating the sample collecting interval.
improving_threshold=0.9, after this fraction of steps only accept graphs that score better.
improving_linear_start=0.3, start punishing worse graphs starting from this fraction of steps.
max_size_diff=20, choose CIPs such that the overall size of a graph does not deviate too much from the start
accept_min_similarity=0.65, we want new graphs to be at least this similar to the last one.
keep_duplicates=False, do not output the same sample twice.

3.2 One class SVM

We have previously seen that the graph generation process is guided by a one class SVM. First we need to find out if the one class SVM is performing well on RNA data.

For this test I choose a number of graphs. Then I randomly select this number of sequences from the set of seed sequences of the RF00005 family. The sequences

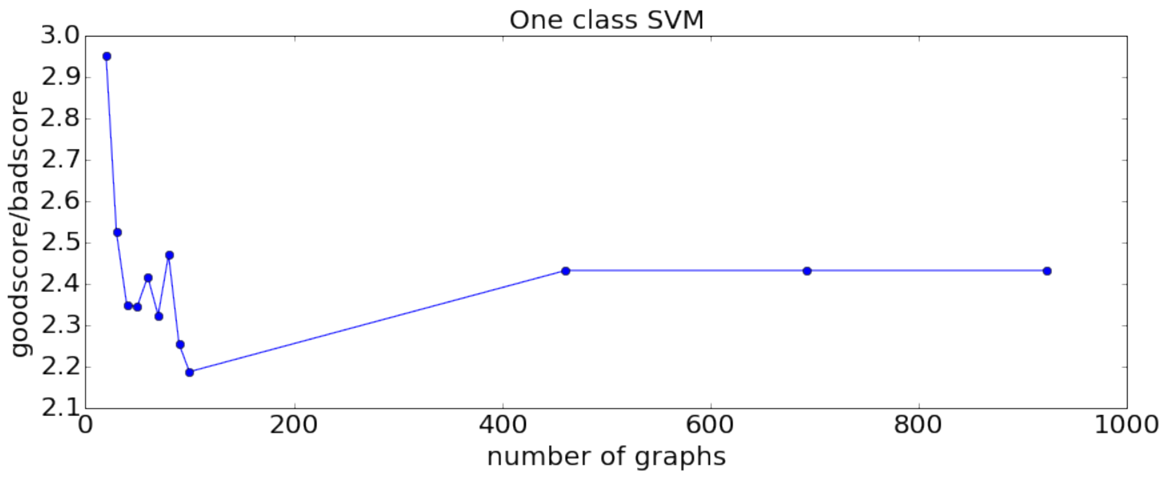

FIGURE 3.1: learning curve for one class SVM

are then folded and a corresponding graph is created. On these graphs the one class SVM is trained.

For each graph in the training set the SVM score is calculated. As described in chapter 2.1.3 the score is in $[0, 1]$. Then I take a closer look at the associated dot bracket string. If it resembles a t-RNA I save the score in a first list l_1 otherwise in a second list l_2. for the lists the arithmetic mean is calculated and the mean of l_1 divided by the mean of l_2 is the final result. E.g. if the result is 2.0 the good scores are on average twice as large as the scores of the non t-RNA like shaped instances. This is done 5 times for each number of graphs, the average is depicted on the y axis of fig 3.1. We observe that the SVM separates graphs by their folding shape efficiently.

As described earlier the kernel software is offering the option of working with a graph minor alongside the original graph. I repeated the last test with this option enabled. The result can be observed in figure 3.2. To my surprise the performance diminishes.

3.3 Covariance model

We generate sequences from differently sized input sets and put them to the covariance test. Tests repeated 3 times. Figure 3.3 shows the result for RF01725. We observe that we are always well above the threshold of 38 log odds score as the generated sequeneces are on average at 51 with a standard deviation (±) of 14. As

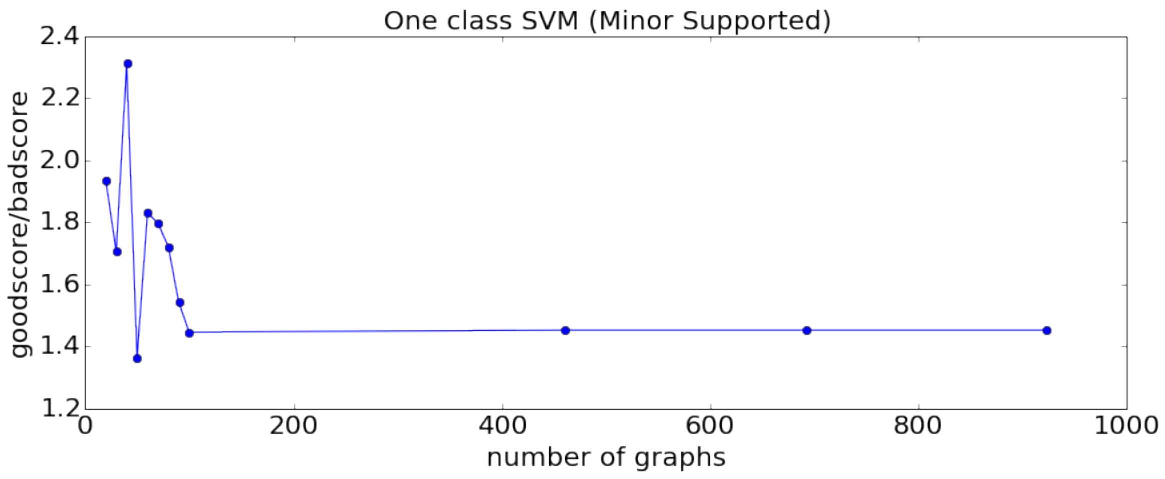

FIGURE 3.2: learning curve for one class SVM supported by graph minors as in 2.2.1

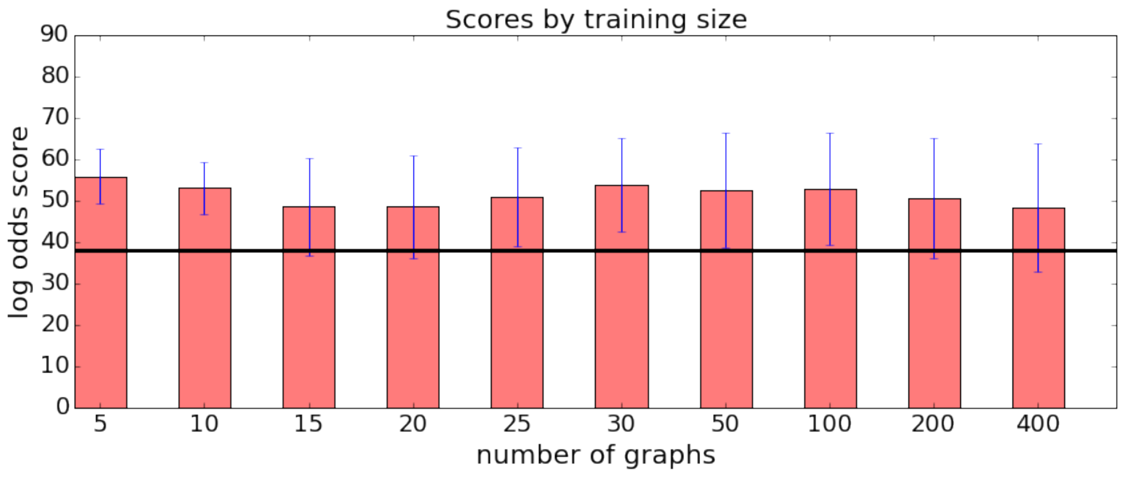

FIGURE 3.3: Sampling of sequences from the RF01752 family. Standard deviation is seen in blue, for any number of input graphs we are well above the threshold of 38 log odds score units

the results look very similar, I am not including a graphical representation for the other families. RF00005 has a threshold of 29 according to Rfam, the generated samples are in 42 ± 20. The cutoff for RF00162 is at 44 while the samples are at 77 ± 12.

3.4 Predictive performance

Two families are observed, on fractions of them samples will be created another fraction is used as test set. The two original sets, the two generated sets and the two mixed sets (original+samples per family) are used to train an SVM each. The

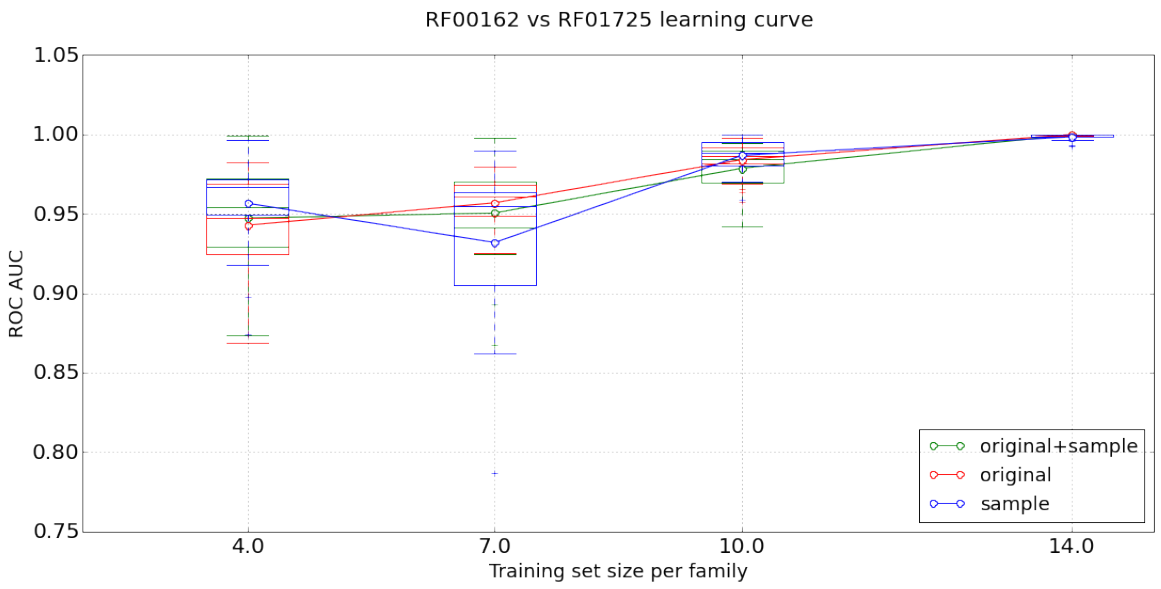

FIGURE 3.4: Sampled from the riboswitch families, we observe the performance of a discriminative machine learning model.

test set is used to estimate the quality of the SVM model. A good model means that meaningful instances were used to train it.

In my test I use 100 sequences from each family, 30 of which will be the test set. Tests run 20 repeats.

The learning curves in figures 3.4 and 3.5 show that the seed sequences are easily separable. The original and the original+sample set perform consistently well. The samples produce comparatively unreliable results by themselves. Considering that the sequence generation procedure is optimized to find a high scoring instance and not an informative one, this could have been expected.

3.5 Summary

First I evaluated the one class SVM. I wanted to know if sequences that were folded in the way that corresponds to the family-structure, are scored better than those which were folded differently. The results show that this is the case, and that this is also the case for very few training instances. Since the SVM is guiding the creation of new sequences, it is important to establish its effectiveness.

Next we made sure that the sequences created by my method are in line with our expectations. The algorithm should create sequences that most likely belong

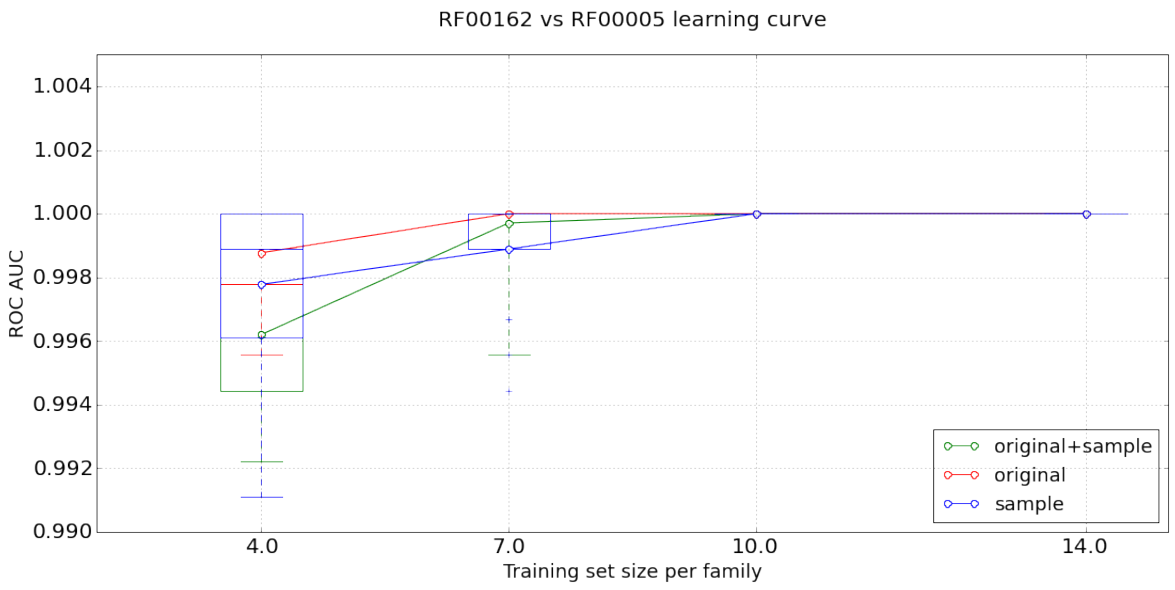

FIGURE 3.5: Discriminative performance for one of the riboswitch families against t-RNA. Already 4 instances are enough to train a reliable discriminative SVM.

to the RNA family the method was trained upon. Because we are working with RNA we can evaluate the resulting sequences by using covariance models, which is a method proven to be reliable. The result is that independent of the number of graphs used to train the method, consistently relevant sequences were created. This is the case for all of the RNA families I worked with. This was the goal of my thesis.

In the last set of tests I demonstrated that good instances are created by a method that does not rely on covariance models. If one is not be working with RNA, covariance models are not available. If one was to apply my method in another problem domain, a test like this would be used to determine the quality of results. In the test, samples of two families were used to train an SVM whose performance was measured. In both tests we saw, that the generated sequences are about as informative as the original sequences.

Chapter 4

Discussion

We have seen that my approach creates convincing instances of RNA sequences. Applying this approach to RNA has advantages and disadvantages. For the RNA problem domain we were able to rely on the Rfam and covariance models to confirm that our produced results were good. Because the structures within a family are not very diverse, I think we were not able to observe the full potential of this approach.

Leaving the RNA problem domain, one could use my approach to improve the sampling of chemical compounds. Costa demonstrated that the base algorithm is compatible with chemical molecules. With my approach one is able to, for example, group atoms that form a cyclic compound and thus the sampling process could be aware of rings. The cycle minor builder is already implemented and ready to be explored.

My strategy for creating graph minor is connected to the efficiency of the procedure applied to RNA. Not in every problem domain an obvious way of building minors is available. Therefore learning how to abstract a graph into a minor would be a great improvement. I implemented an example procedure that creates minors by contracting nodes based on their influence on the SVM model. Based on this implementation an efficient generic minor learner might be built.

In the procedure I presented the core of the CIP always consisted of the nodes that were contracted in core nodes of the minor graph. We named this $C_R^v(G', G)$. This resulted in very large chunks of an RNA molecule being altered at a time. Please be reminded that G' refers to the minor graph while G is the original. In my algorithm we used $C_R^v(G', G)$ as the core that is swapped, $I_{R,B}^v(G', G)$ as the interface and when calculating the hash of this interface we also included $I_{R,T}^v(G')$. During sampling one could also make smaller changes by using productions that use $C_{R'}^v(G)$ as a core with $I_{R',B}^v(G)$ as interface and hash the interface again with $I_{R,T}^v(G')$. Notice that I just introduced R' which is a radius for the base graph. These small and bigger substitution types could be used together in the same sampling session. The basic functionality is already part of my software.

Chapter 5

Conclusion

I have presented a method that is clearly able to learn and construct RNA molecules.

I did so by adapting a sampling method for graphs by Costa [1]. By my method long range dependencies in graphs become depictable and core graphs become more flexible, as they were simple neighborhood graphs with a radius before. This is also the first time the grammar is used with directed graphs. In chapter 4 and section I mentioned possible modifications to the method.

We had to give up MH sampling, but that was largely due to the refolding of the sequences.

When adapting this method to other problem domains, a hindrance might be that one has to provide a method to calculate graph minors. Once this is available though, graph minors can improve the capabilities of grammar based sampling approaches.

Appendix A

Appendix

A.1 Used Software

MUSCLE v3.7
RNAalifold 2.1.9
Rfam 12.0
Python 2.7
numpy 1.8.0
scipy 0.14.0
scikit-learn 0.17.0
networkx 1.10
matplotlib 1.5
pygraphviz 1.3.1
EDeN (Dec 10 2015)

Bibliography

[1] F. Costa and D. Sorescu. The constructive learning problem: an efficient approach for hypergraphs. *Workshop on Constructive Machine Learning (CML) NIPS, Lake Tahoe, Nevada, USA, 10 December*, 2013.

[2] M. Zuker. Prediction of RNA secondary structure by energy minimization. *Methods Mol. Biol.*, 25:267–294, 1994.

[3] Anke Busch and Rolf Backofen. INFO-RNA –a fast approach to inverse RNA folding. *Structural bioinformatics*, 22:1823–1831, 2006.

[4] M. Marshall S. Griffiths-Jones, A. Bateman. Rfam: an rna family database. *Nucleic Acids Res.*, 31:439–441, 2003.

[5] Metropolis N., A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller and E. Teller. Equations of state calculations by fast computing machines. *Journal of Chemical Physics*, 21:1087–1092, 1953.

[6] M. P. Vecchi S. Kirkpatrick, C. D. Gelatt. Optimization by simulated annealing. *Science*, 220:671–680, 1983.

[7] F. Costa and K. De Grave. Fast neighborhood subgraph pairwise distance kernel. *Proceedings of the 26th International Conference on MachineLearning*, 22:255–262, 2010.

[8] I. Damgåd. A design principle for hash functions. *Advances in Cryptology–CRYPTO89 Proceedings, Springer*, 22:416–427, 1990.

[9] J.Shawe-Taylor, A. J. Smola, R. C. Williamson B. Schölkopf, J. C. Platt. Estimating the support of a high-dimensional distribution. *Neural computation*, 13:1443–1471, 2001.

[10] et al. J. Platt. Probabilistic outputs for support vector machines and comparisons to regularized likelihood methods. *Advances in large margin classifiers*, 10:61–74, 1999.

[11] H. Ehrig G. Rozenberg. Handbook of graph grammars and computing by graph transformation. *World Scientific*, 11:0–0, 1999.

[12] P. Kohvaei F. Costa, R. Kleinkauf. Rnasynth: a graph kernel approach to learn constraints for rna inverse folding. *Constructive Machine Learning Workshop ICML 2015*, 0:0–0, 2015.

[13] Robert C. Edgar. Muscle: multiple sequence alignment with high accuracy and high throughput. *Nucleic Acids Res.*, 32:1792–1797, 2004.

[14] Ivo L. Hofacker Stephan H. Bernhart. Rnaalifold: improved consensus structure prediction for rna alignments. *BMC Bioinformatics*, 9:474, 2008.

[15] S. R. Eddy E. P. Nawrocki. Infernal 1.1: 100-fold faster rna homology searches. *Bioinformatics*, 29:2933–2935, 2013.