Master's Thesis

Reinforcement learning techniques in RNA inverse folding

Parastou Kohvaei

August 2015

Albert-Ludwigs Universität Freiburg
Department of Computer Science
Chair of Bioinformatics

Candidate
Parastou Kohvaei
Matr. number
3210422
Working period
15. 01. 2015 – 26. 08. 2015
Reviewers
Prof. Dr. Rolf Backofen
Dr. Frank Hutter
Supervisor
Fabrizio Costa

Abstract

A non-coding RNA molecule functionality depends on its structure, which in turn, is determined by the specific arrangement of its nucleotides. The inverse folding of an RNA refers to the problem of designing an RNA sequence which will fold into a desired structure. This is a computationally complex problem. Algorithms which solve this problem take different approaches, but they share the following attitude: They start from an initial sequence or population and try to move it towards a desired product by performing normal or optimized search methods. RNA inverse folding programs are given different constraints such as GC-content ranges or basepair or nucleotide configurations. The output is normally one or more sequences which fold to the target structure.

This work introduces a basic system that given a set of sample RNA secondary structures, produces models which generate structures similar to the sample set. The objectives and constraints are automatically extracted from samples. For doing this, a system is designed which generates models by performing learning on families of RNA sequences. This system consists of two subsystems: one responsible for decomposing secondary structures of sample RNAs into structural features and building a structural features corpus. It also extracts neighborhood connectivity models of structural features in the form of N-grams. The other subsystem is a reinforcement learning framework which uses the corpus and connectivity rules to produce models for generating structures which are similar to the samples.

Results in this work show that the current system is able to produce models from RNA families which have a symmetric shape. To make the system capable of dealing with a broader range of RNA families and producing structures with functionalities identical to the sample structures, a refined feature extraction module has been added to the system. This module extracts the GC-content, size and local information of structural features and builds a refined feature corpus. This can provide the basis for a new set of experiments and a start point for producing models with practical applications.

Kurzfassung

Die Funktionalität eines RNA Moleküls hängt von seiner Struktur ab, die wiederum durch seine spezifische Anordnung seiner Nukleotide bestimmt wird.Inverse RNA-Faltung bezeichnet das Problem eine RNA-Sequenz zu gestalten, die sich in eine gewünschte Struktur falten wird. Es ist ein Problem mit hoher Rechenkomplexität.Algorithmen, die dieses Problem lösen, benutzen verschiedene Ansätze, aber sie teilen einen Grundansatz: Sie beginnen mit einer Initialsequenz oder Initialpopulation und versuchen diese zu einer gewünschten Sequenz oder Population zu transformieren, indem sie normale oder optimierte Suchmethoden benutzen.RNA inverse Faltungsalgorithmen können ihren Suchraum zum Beispiel durch bestimmte GC-Gehälter oder durch strukturelle und sequenzielle Randbedingungen eingrenzen.Die Ausgabe ist normalerweise eine oder mehrere Sequenzen welche in die gewünschte bestimmten Struktur falten.
Diese Arbeit stellt einen Algorithmus vor, der basierend auf Beispiel RNA sekundär Strukturen Modelle erstellt,welche gleichartige Strukturen generieren können.Die Ziele und Beschränkungen des Algorithmus werden automatisch extrahiert aus einer Menge von RNA Beispielsequenzen. Um dies zu erreichen, gestalten wir ein System, das Modelle generiert, indem es auf funktionalen Familien von RNA Sequenzen lernt.Der Algorithmus besteht aus zwei Teilsystemen. Das erste Teilsystem zerlegt die gegeben sekundär Strukturen in ihre strukturellen Merkmale und formt daher den strukturellen Merkmalskorpus.Es extrahiert Nachbaraschafts-Zusammenhangs-Modelle der strukturellen Merkmale in der Form von „N-grams“.Das zweite Teilsystem bildet der Reinforcement Learning Rahmen. Hier werden die Informationen des Merkmalskorpus und die Zusammenhangregeln zur Generierung des Modells genutzt, welche gleichartige Strukturen erstellt.
Ergebnisse diese Arbeite zeigen, dass der Algorithmus in der Lage ist Modelle basierend auf RNA Familien mit symmetrischen Struktur zu generieren.Des weiteren wurde der Algorithmus mit einem verfeinert Merkmals-Extraktions-Model ausgestattet um mit einer größeren Anzahl von RNA Familien umzugehen und sicherzustellen, dass die generierten Sequenzen

die gleichen Funktion wie ihre Beispiel Strukturen besitzt. Dieses Modell extrahiert den GC-Gehalt, Größen- und Ortsinformation der strukturellen Merkmale und formt den verfeinerten Merkmalskorpus. Es kann als Ausgangspunk für neue Experimente genutzt werden um Modelle mit spezifischen Anwendungen zu produzieren.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

An RNA molecule functionality depends on its structure, which in turn, is determined by the specific arrangement of the nucleotides in the RNA molecule. The inverse folding of an RNA refers to the problem of designing a RNA sequence which will fold into a desired structure. This is a computationally complex problem.

Algorithms which solve this problem take different approaches, but they share the following attitude: They start from an initial sequence or population and try to move it towards a desired sequence or population by performing normal or enhanced search methods.

These algorithms mainly use the following approaches for solving the problem: dynamic programming, genetic algorithms, and constraint satisfaction methods. Many of RNA inverse folding programs accept various explicit constraints which define the characteristics of the desired target structure. These constraints are in the form of GC-content ranges, basepair or nucleotide conservations etc. The final product is normally one or more sequences which have a specific structure and hence, a desired functionality.

In this work we introduce a system for producing models that generate structures similar to the representative structure of a given RNA family. The objectives and constraints are automatically extracted from the sample set. This system has been designed and implemeted in two phases:First a standalone system for decomposing RNA structures into structural features have been built. A second system, a reinforcement learning framework, has

been then built and integrated with the first system. We have called it *RL-RNA*.
Results from the first set of experiments show that this system is able to capture the coarse shape of simple structures. In order to provide the system with fine-grained information of sample structures, another module has been added to the system. It extracts GC-content and size and relative location information of structural features in a given secondary structure. The feature extraction mechanism is not limited to the three measures mentioned above. Other measures could be introduced and incorporated to this module.

1.1 Non-coding RNAs

1.1.1 Biological view

RNA

Ribonucleic acid (RNA) is a macro molecule which plays an important role in protein synthesis in biological cells. RNAs consist of long chains of chemical building blocks called nucleotides which are compounds containing a sugar, phosphate groups, and a nitrogenous base. The four nitrogenous bases in nucleotides are adenine, guanine, cytosine, and uracil which are identified by A, G, C, and U letters respectively.

ncRNA

Non-coding RNA is any RNA sequence which does not encode a protein. There are different groups of ncRNAs with various roles in cellular processes. Recent advancements in genomics have shown that in complex biological organisms, a major part of the genetic information is copied into none-coding RNAs.[16] This has brought a lot of attention to the study of structure and functions of different types of non-coding RNAs.

RNA secondary structure

RNA molecules fold into complex structures by forming bonds between pairs of G and C, A and U, and A and G bases. These are called secondary structures which convey the functionality and purpose of the RNA. Structurally

related RNA sequences belong to the same family. There are two main problems which are related to the secondary structure: RNA folding and inverse folding.

Secondary structure features

RNA secondary structure is assembled from a number of structural features. These basic building blocks are repeating in different numbers and combinations to form the unique structural and functional characteristics of the sequence. These features are as follows:

- stack (stem)
- internal loop
- multiloop
- hairpin loop
- bulge
- dangling end

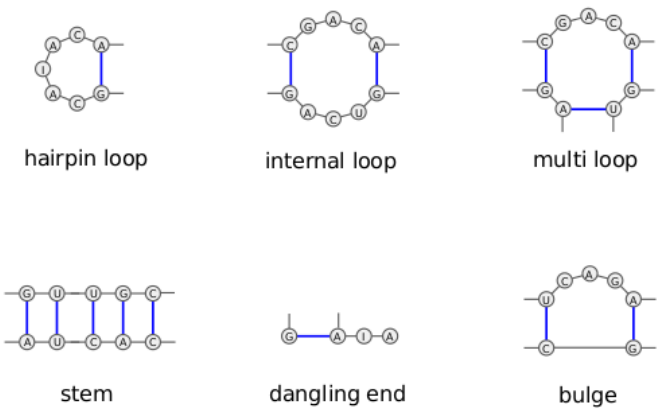

Figure 1.1: RNA secondary structure features

1.1.2 Computational aspects

RNA secondary structure prediction

Also known as RNA folding prediction is one of the classical problems in computational biology which deals with predicting the most likely secondary structure given an RNA sequence. There are many different methods for solving this problem which boil down into comparative, thermodynamic, and probabilistic approaches.

Multiple sequence alignment

Is the multiple alignment of two or more biological sequences to capture their common motifs and conserved regions. The results of alignments are very useful in structure prediction and optimized searches in genomics databases. [25]

Consensus secondary structure

Is the representative structure for a group of related biological sequences. In the case of RNA, the consensus secondary structure of a family mainly deals with basepair conservations.

Covariance model

Is a probabilistic model that can generate representative members of an RNA family. It captures characteristics of a multiple sequence alignment in both nucleotide and pairwise consensus structure aspects. CMs are generalizations of Hidden Markov Models and are produced from annotated representative members (seeds) of a family. They can automatically annotate single sequences to decide if they are related to the family. In this case a covariance score is calculated for the given sequence. [6]

1.2 Synthetic biology and RNA inverse folding

Here we give an introduction to *synthetic biology* as one of the computational fields in biology and we will talk about *RNA inverse folding*, one of the most

famous problems in this field. We will give an overview on the computational complexity of this problem and state-of-the-art algorithms for solving this problem.

1.2.1 Synthetic biology and its applications

Synthetic biology is a broad interdisciplinary field of science which combines several disciplines such as biotechnology, systems biology, and computer science and is highly related to genetic engineering.[8] Synthetic biology involves the engineering and synthesis of biological systems with functions which do not already exist in nature. These systems might range from a single molecule to an entire organism. This field of study is rapidly growing and offers a very diverse spectrum of research projects. There are several important application areas into which, synthetic biology will bring along promising changes, some of which are:

- Biomedicine
- Synthesis of biopharmaceuticals
- Sustainable chemical industry
- Environment and energy
- Production of smart materials and biomaterials

1.2.2 The RNA Inverse Folding Problem

RNA inverse folding is the problem of finding one or more sequences which will fold into a specific secondary structure. This problem, which belongs to the field of synthetic biology, has several application areas: Designing non-coding RNAs, which are involved in gene regulation, chromosome replication, and RNA modification. Construct ribozymes and riboswitches, which may be used as drugs and therapeutic agents in research. Building self-assembling structures from small RNA molecules, which is used in nanobiotechnology. [9]

1.2.3 Hardness of the Problem

The problem of designing RNA sequences can be reformulated into a Hidden Markov Model (or a Stochastic Context-free Grammar) using the probabilistic formalism. It is then proved to be NP-hard. [22]
This means that finding a global solution would require exponential time. In addition, many of the existing RNA inverse folding methods use some folding algorithm at some point. Given that the folding problem is NP-complete [2] , the complexity of the real problem is even higher. Introducing efficient heuristics to break down this complexity drives many researchers to look for new approaches to solve this problem.

1.3 Algorithms for RNA inverse folding

There are several algorithms for solving the problem of inverse folding. They take different approaches and accept various constraints and use some heuristics. Here we introduce some of the most known algorithms and make a brief explanation about how they work.
The first three algorithms use the folding function of Vienna RNA Package as the folding problem solver. Frnakenstein and ERD also use RNAfold. Modena can use different problem solvers such as RNAfold and CentoidFold.

RNAInverse (1994)

Uses dynamic programming. Uses base pairing matrices of the partition function as heuristics. [11]

RNA-SSD (2004)

Uses constraint satisfaction method. Uses probabilistic sequence initialization heuristics. [1]

Info-RNA (2006)

Is a two-step algorithm which uses dynamic programming as the initial step and stochastic local search as heuristics. [3]

Modena (2011)

Uses a genetic algorithm in combination with multi-objective optimization and outputs several optimal solutions per run. It accepts multiple objective functions as constraints. [24]

Frnakenstein (2012)

Uses genetic algorithms and utilizes local search (adaptive walk) as heuristics. It can find multiple target structures. [15]

incaRNAtion (2013)

. Uses a probabilistic model (weighted sampling) and fixed constraints. Has low space and time complexity in comparison to other methods. [20]

RNAiFold (2013)

Uses a non-heuristic constraint satisfaction method and outputs the whole target space. [10]

ERD (2013)

Takes the genetic approach in combination with hierarchical decomposition of secondary structures as heuristics. [7]

RNAdesign (2013)

Uses graph coloring in combination with local optimization and outputs multiple target sequences. [12]

antaRNA (2015)

Applies ant colony optimization technique on multi-objective constraint declarations, it introduces multiple target GC specifications and fuzzy structure constraints. [13]

1.4 Machine learning in bioinformatics

One of most promising branches of artificial intelligence in research and industry is machine learning. Its applications span over computer vision, object recognition, robotics, data mining, and many other practical fields. In this section, we talk about the applications of machine learning in bioinformatics.

1.4.1 Artificial Intelligence and Machine Learning

Artificial intelligence is the art of making intelligent machines which can be used in tasks that require cognitive abilities. Knowledge plays a central role in artificial intelligence, and designing systems that can acquire new knowledge from data is one of the main goals of artificial intelligence. [21] Machine learning is the study of designing computer algorithms which can automatically improve their performance by learning from experience data. It is one of the most practiced subfields of artificial intelligence and its usage in different industrial and research areas is growing fast. [17]

1.4.2 Types of machine learning

Machine learning techniques are categorized into three major fields: supervised learning, unsupervised learning, and reinforcement learning. Here a brief introduction is given to the two first fields and in the following section, we introduce reinforcement learning in more detail.

Supervised learning

In supervised learning, the system is introduced to some labeled data. The task of the system is then to learn the hypothesis which best represents the correlations between the data and labels. It can then predict the labels of new data which were not seen before. Supervised learning can be applied to predicting discrete labels (classification) as well as continuous ranges (regression). There are also more advanced methods which combine both approaches.

Unsupervised learning

The most famous example of unsupervised learning is clustering. Given a non-annotated data set, a clustering method tries to find similarities and recurring patterns among the members of the set and group the data based on these similarities.

1.4.3 Biological data revolution

Biological data has two major characteristics: it is complex and huge. This body of data has a fast pace of growing. This has caused many traditional computational techniques to fail to analyze and manage the big data. The need for adaptive systems which are capable of dealing with large data sets gave rise to machine learning techniques which are now vastly used in different biological application areas.

1.4.4 Machine learning applications in bioinformatics

There are several domains of biology which benefit the most from machine learning methods. Genomics, proteomics, system biology, evolutionary biology, synthetic biology and biological data management are some examples. [14]

Classification

Supervised learning has different applications in bioinformatics, here we mention some:

- predicting protein secondary structure with Artificial Neural Networks
- RNA gene finding using Support Vector Machines
- identifying genes using classification trees
- predicting RNA secondary structure with KNN classifiers

Clustering

Clustering methods are vastly used in microArray analysis.

1.5 Reinforcement learning

In this chapter a theoretical introduction to reinforcement learning is given. The mathematical foundation of reinforcement learning is briefly presented at first. In the rest of the chapter, one of the most popular and flexible approaches of reinforcement learning called Q-learning is introduced.

1.5.1 Classical reinforcement learning

Reinforcement learning is learning by experience. In this sense, it is different from the other two fundamental approaches of supervised learning and unsupervised learning.
In reinforcement learning, an *agent* learns to achieve an objective while interacting with an *environment*. The environment is the embodiment of a specific learning problem. It consists of a set of different states S and a set of actions A.
The agent selects an action at each state, is transferred into a new state and gets a reinforcement signal in the form of a reward or punishment. The transition from one state to another might be a one-to-one mapping (deterministic) or a one-to-many mapping (stochastic). In the second case, we use probabilistic transition functions. [23]

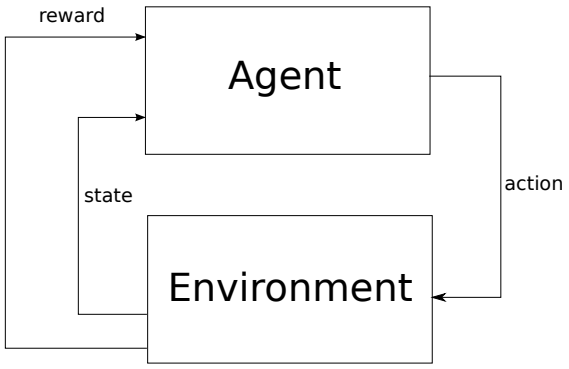

Figure 1.2: Agent-environment interaction in reinforcement learning

1.5.2 Markov Decision Processes

A state is said to have *Markov property* if it retains all the necessary information about past experiences so that the agent does not need to know about the history of its actions. A reinforcement learning task with Markov states, can be fully represented by the current state, current action, and the reward to this action. In this case, we call the task a *Markov decision process* or for short a MDP.
A *finite MDP* is a MDP with finite state and action sets. The transition probabilities of any given state s and action a for possible next states ś in a finite MDP are calculated by the following formula.

$$\mathcal{P}^a_{ss'} = Pr\left\{s_{t+1}=s' \mid s_t=s, a_t=a\right\}.$$

Figure 1.3: Markov decision process transition function (figure from [23])

1.5.3 Reinforcement learning elements

Agent

A learning agent interacts with the environment in discrete time steps. At each step it selects an action which causes a transition to the next state and brings back a reward. Through time, the agent will learn to select actions which maximize the accumulated reward . Depending on the system architecture, the learning and decision making mechanism could be implemented in the same or different structures. Actor-critic methods fall into the second group.

Environment

Environment defines the characteristics of each specific task which is to be formulate as a reinforcement learning problem. A well defined problem which can guarantee success needs to capture all the relevant features of the task at hand and reformulate them into environment dynamics which are usually the transitions between states and the objective function.

Policy

Policy is the mapping from states to actions. A policy π enables the agent to decide which action to pick at a certain time step and in a certain state. The learning process is all about finding the optimal policy in the policy space of a problem. An optimal policy is the policy which achieves the maximum long term payback. The optimal policy might or might not be unique.

Objective function

The goal of a agent during each step of learning is to maximize an *objective function*. This objective function normally consists of two parts: an immediate *reward* r that the agent receives at each time step by selecting actions and making transitions to other states, and a *state value* v which specifies the amount of expected accumulated reward when the agent continues from the next selected state. Value and reward functions are two main directives for guiding an agent towards the best course of action.
Figure 1.4 shows the value function for a Markov decision process in a given state s, if the agent starts from this state and follows a policy π.

$$V^{\pi}(s) = E_{\pi}\{R_t | s_t = s\} = E_{\pi}\left\{ \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \;\middle|\; s_t = s \right\},$$

Figure 1.4: Value function for state s (figure from [23])

Delayed reward

In many real world situations, such as games or control problems, it is not possible to credit each time step with an immediate reward. In such cases the system uses delayed rewards which is a final total credit at the end of each experience trial. The step-wise rewards are then formed through time by the final credit being back-propagated through the whole state sequence.

Exploration vs. exploitation

In order to avoid local maxima in the search for the best state value, the agent has to make a balance between following the current policy (exploita-

tion) and trying actions which will lead to unvisited states (exploration). One of the common approaches to this problem is ϵ-greedy action selection method. This approach will choose the best action with probability $1 - \epsilon$ or a random action with probability ϵ.

1.5.4 Model-Based vs. model-free Learning

A model of an environment is the mapping from current state and the action taken at that state to the next state and the reward returned for this transition. Since in many real world problems the model is not known, classical methods which need a model to operate would fail. In such cases *model-free* approaches are used.
A model-free learning algorithm uses sample sequences of states. These samples might be produced by experience or through simulation. *Monte Carlo* and *temporal difference learning* methods are the two main classes of model-free learning approaches. *Q-learning* is a popular yet simple temporal difference algorithm which is used as the main learning algorithm in this thesis. Here we give an introduction to this algorithm and present its formal notation.

Q-learning

Q-learning has a vast application in real-life situations in which, not all of the necessary assumptions for the theoretical approach are fulfilled. In many practical applications where transition functions of the environment are unknown or hard to capture, we can use Q-learning.
As a model-free approach, Q-learning uses *action-value* function instead of the usual value function.

Action-value function estimation

Given a policy π, an action-value q of an action a in a state s specifies the expected reward of choosing a at s and continuing by following π from the next state. Figure 1.5 demonstrates the iterative formula for updating the action-value function Q.

It is a one-step update, meaning that action-values at each states are influenced by the values of the immediate next states.

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha\Big[r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)\Big].$$

Figure 1.5: One-step Q-learning action-value function estimation (figure from [23])

Algorithm

Q-learning starts with random guesses about action-values of states and iteratively updates these values by going through learning episodes. If all action-values converge, the learning finishes and the mapping from states to their respective action-values is returned as the policy.

Initialize $Q(s, a)$ arbitrarily
Repeat (for each episode):
  Initialize s
  Repeat (for each step of episode):
    Choose a from s using policy derived from Q (e.g., ε-greedy)
    Take action a, observe r, s'
    $Q(s, a) \leftarrow Q(s, a) + \alpha\big[r + \gamma \max_{a'} Q(s', a') - Q(s, a)\big]$
    $s \leftarrow s'$;
  until s is terminal

Figure 1.6: One-step Q-learning algorithm (figure from [23])

1.6 Thesis contribution

This work contains the design and implementation of a new system which generates RNA structural feature databases in one hand and learns models capable of producing similar RNA structures using these features on the other hand. Two versions of the systems are introduced here: the first version is able to learn coarse shape of RNA structures. The main focus of the experiments is on this version. The second version is an improvement designed to automatically extract detailed information from structural features

in the database and incorporate that information to the learning process. Experiments on the second version are a subject of future work.
In the next chapter we provide the reader with information about design and implementation of both versions of the system. In chapter 3 we give all experimental setups necessary for reproducing the experiments and show the results of our experiments. In chapter 4 we discuss results and make suggestions for improving the system to turn it to a production tool in real applications; future work is inspired by these suggestions.

Chapter 2

Method

2.1 Formulation

2.1.1 Definition

We want to produce RNA secondary structures which follow the structural rules same to the members of a sample RNA family. Our proposed approach is to identify and obtain meaningful substructures in sample RNAs. We train a learning model to recombine these substructures to produce new structures. The synthesized structures should have same shape and similar substructures as what is observed in the sample set.

2.1.2 Encoding

We use a decomposition scheme which is based on identifying structural features in the secondary structure of a sequence. We use a procedure to decompose all the sequences in the target family and build a structural feature base.

A tool suit for connecting structural features and producing valid combinations has been designed and implemented. Decomposition and recombination operations involve graph representations of secondary structures of RNA sequences. A set of rules are inferred from the consensus models of each family which govern combination operations (grammar). The extracted database and grammar are then used in a learning system to learn policies for generating combinations which comply with the family grammar.

2.1.3 System outline

To build a system which can solve this problem, two main sub-systems are implemented:

- A system for generating the structural feature base and maintaining the grammar
- A learning system

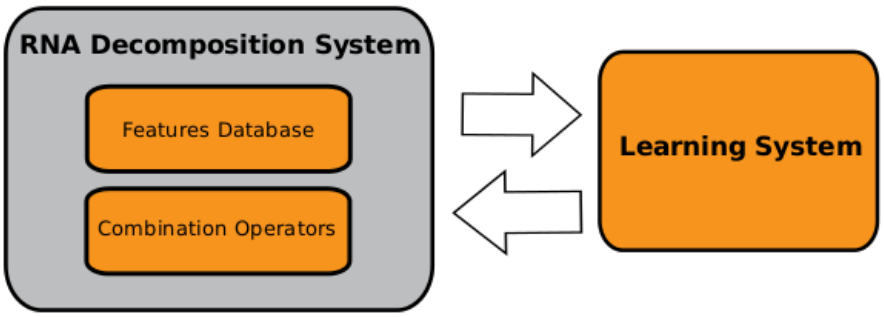

Figure 2.1: A schematic view of the system outline

2.1.4 Technical notes

For implementing the system and running the experiments, we use the following programs and data sources:

Programming language

Python programming language is used for implementation.

RNA structure representation

To present RNA secondary structures and structural features in the form of graphs, networkx module in python is used. EdeN tool suit is used to down-

load sample datasets in the form of fasta files, pre-process fasta sequences and generate mfe RNA secondary structures in the form of networkx graphs, and visualize intermediate and end results. [5]

Experimental data

Experimental data are RNA families from the Rfam database. [18] Each family in Rfam database is set of non-coding RNA sequences which is represented by manually curated alignments and consensus nucleotide-wise, pairwise, and structure-wise models of the family.

2.2 RNA coarse shape learning

The aim of this part of the methodology is to test if a reinforcement learning system can correctly learn the most typical shape of the secondary structure of an RNA family. The system learns to combine secondary structure features and build up structures which look like the "consensus secondary structure" of an Rfam group.

2.2.1 Problem encoding

Given a graph which represents the secondary structure of an RNA sequence, a series of operations is done to analyze and decompose the graph.

2.2.2 RNA secondary structure decomposition

Finding break points

A break point in the graph is the attaching point of a stem to another structural feature. Spotting break points in a secondary structure is highly dependent on the representation of the structure. Here we use networkx graphs, so a break point is detected whenever there is a change in the type of connections between nodes from backbones to basepairs or vice versa. These pairs are tagged then to be further processed.

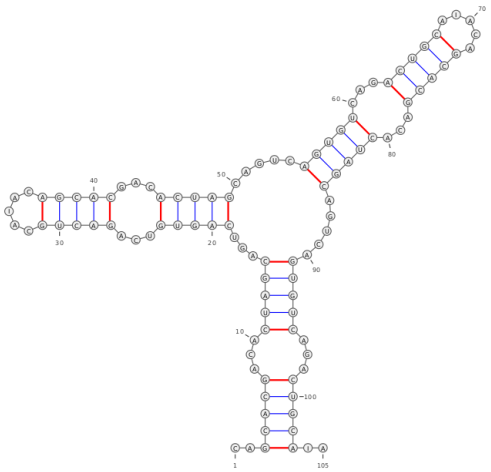

Figure 2.2: Break points in a sample secondary structure

Cutting the graph

The graph is then cut at break points. This yields a set of disconnected components which should be identified and prepared for the next level of operations. The logic implies that all non-stem structures already have a special marking at cut points. We call these *magnetic ends* which are the attach points at the combination stage. Each magnetic end is basically a base pair in the original graph. During the decomposition process, magnetic ends are marked with a different edge label.

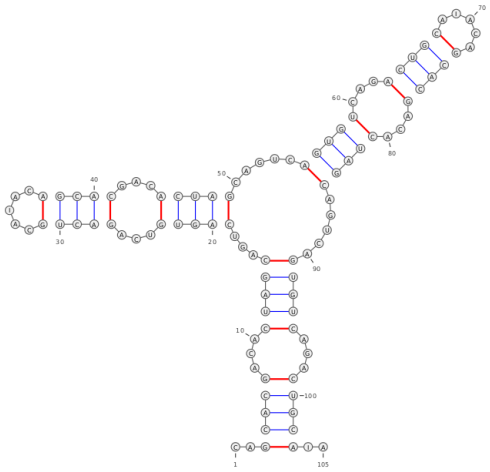

Figure 2.3: Secondary structure cut at break points

Identifying and tagging stems

At this stage, stems are identified and their magnetic ends are detected and marked.

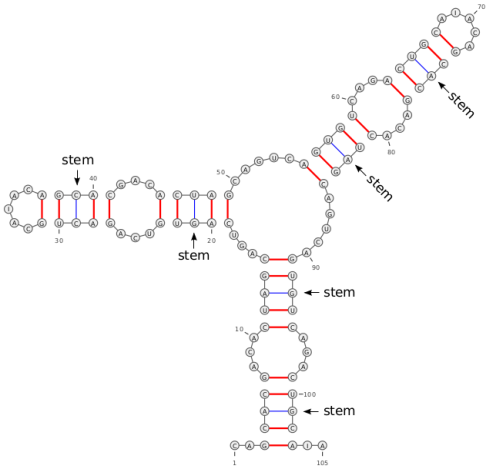

Figure 2.4: Identifying stems

Identifying other structural features

From this point, identification of the rest of the features is straight forward. The current version of the program is capable of identifying internal loops,

bulges, multi loops with 3, 4, and 5 entries, dangling ends, and hairpin loops. It is also possible to identify compound structures such as the combination of a bulge and a hairpin loop, but decomposing compound structures is still not a part of the system. Therefore all compound structures are eliminated from the final structure pool.

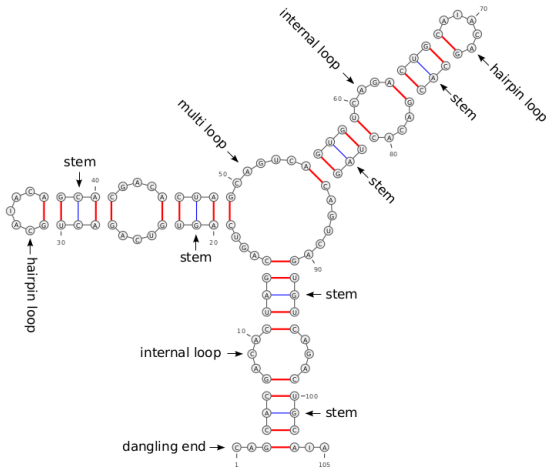

Figure 2.5: Identifying loops and dangling ends

2.2.3 Combining RNA structural features

The aim of this group of operations is to provide the system with a correct way of recombining structural features into bigger structures which may ultimately look and act like a RNA secondary structure. Here we explain the mechanism and talk about some considerations.

Combining two structural features

In our current implementation, we simply create a pair of edges each connecting one node in a magnetic end of a component to a node in the magnetic end of the other component.

A structural feature might have more than one magnetic end (e.g. multi loop with 4 entries). For combination, our program does not consider any priority on choosing one of these ends and throughout the whole set of experiences, the selection is done by randomly picking one of the ends of the component.

Combination order

When dealing with the shape of a secondary structure, the order of the nucleotides in a structural component has no importance. But if we have a focus on the functionality of the resulting combined graph, the order becomes important.
We added a code snippet to our combination function which orders the two nodes in a magnetic end based on their 5-prime order in the original sequence. The combination function then uses this information to connect the nodes with the same order ranking from the two magnetic ends together.

Resetting the magnetic ends

When two features are connected to each other, those magnetic ends which are now connected get *reset*: their edge labels are changed back to **base-pair**; thereafter no more combinations are possible at these ends and we can consider the final RNA sequence to be *end-to-end* connected at these points.

2.2.4 Structural representation and bigrams

Every machine learning algorithm needs some assumptions or "heuristics" which are domain specific and which make the algorithm specifically efficient and successful in finding an optimal solution model for the problem. Reinforcement learning algorithms are no exception. For learning the coarse shape of RNA families, we used the concept of N-grams as heuristics.[4]

N-grams

N-gram is a definition which comes from computational linguistics. A N-gram is a slice of size N from a larger sequence of symbols or characters. In language processing, N-grams have a wide usage in statistical analysis of sentences in a language in order to derive grammatical rules and generate valid sentences or phrases of the language. They have use cases in other fields such as bioinformatics.

Unigrams and bigrams

Unigrams are N-grams of size one and bigrams are N-grams of size two.

2.2.5 Bigram representation of RNA secondary structure

To represent the secondary structure of an RNA sequence as bigrams, we need to find all two-component phrases which can be derived from the graph representation of the structure. Here we bring an example:
In a structure with three hairpin loops, there are naturally three phrases of the form *"stem - hairpin loop"* which show the bigram constellation related to these loops in this graph. Note that there is a distinction between *"stem - hairpin loop "* and *"hairpin loop - stem "* bigrams.
In our system, this distinction is cultivated to emphasize on the order of the combination of structural components. In this sense, *"hairpin loop - stem "* is an impossible pair since a hairpin loop is a closing point in the graph and does not allow further combinations.

Table 2.1: Bigram representation of figure 2.2

Bigram	Occurrence rate
dangling end – stem	1
stem – internal loop	3
internal loop – stem	3
stem – hairpin loop	2
stem – multi loop	1
multi loop – stem	2

2.2.6 Structural feature database

As stated in section 2.1.4, the first stage of the experiment involves decomposing RNA secondary structure graphs to build up a collection of categorized subgraphs. Decomposition stage is a prerequisite for initializing the learning subsystem. There are two main steps at this stage:

Extracting structural features

In RNA coarse shape learning, we use the 8 basic structural features introduced in 1.1.4 to decompose each folded RNA sequence in the family into its features and build a small dictionary with keys as feature names and values as lists containing the actual subgraphs of type "feature". Figure 2.6 and figure 2.7 show a sample RNA secondary structure before and after decomposition into its structural features.

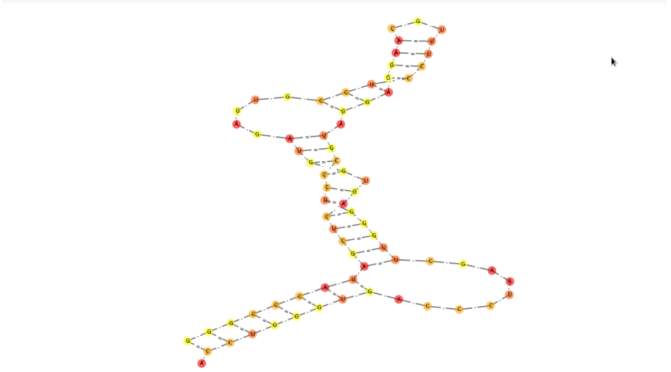

Figure 2.6: Secondary structure of a sample tRNA

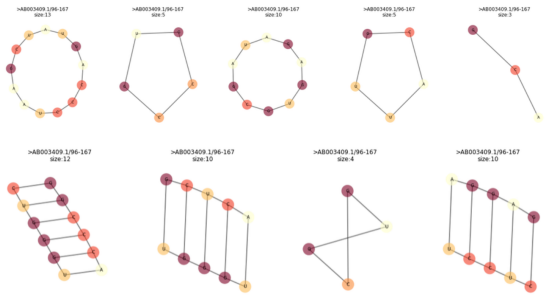

Figure 2.7: Decomposed sample tRNA

Structural feature corpus - RNA family corpus

At this step, all dictionaries are merged into one general dictionary with the same structure and same keys. We will call it the *feature corpus*. Feature corpus is used to setup the learning system. It is also used during learning trials.

In coarse shape learning, we refer to the feature corpus as *family corpus*. It is a dictionary which keeps all the basic structural features found in a RNA family in distinctive chapters. Family corpus is the base of many feature and rule extraction operations. Figure 2.8 shows a graphical demonstration of three different chapters of family corpus of the tRNA.

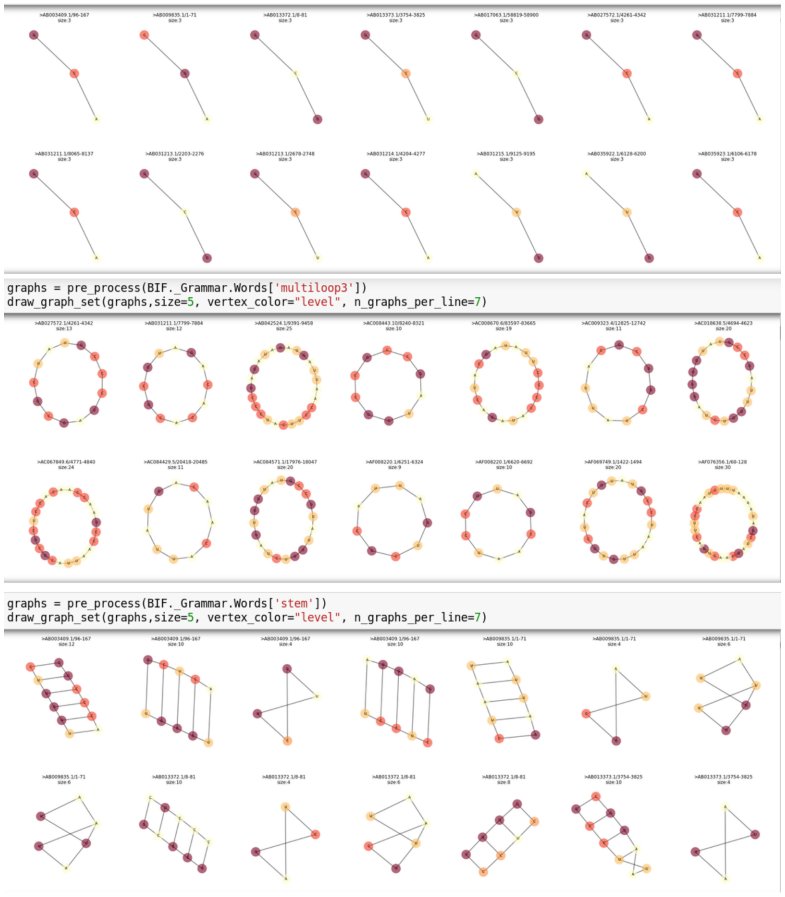

Figure 2.8: tRNA family corpus

2.2.7 Grammar

In order to prepare the system for learning, some basic rules must be inferred from the sample set. Bigram representation of RNA secondary structures is the main context for defining basic rules on what and how to combine. For RNA coarse shape learning, two sets of operations have been used:

Extracting bigram set of consensus secondary structure

The system extracts the building bigrams of the consensus secondary structure. This bigram representation is used in the learning subsystem as a prototype for producing new structures. Figure 2.9 and table 2.2 show a sample consensus structure and its bigram representation respectively.

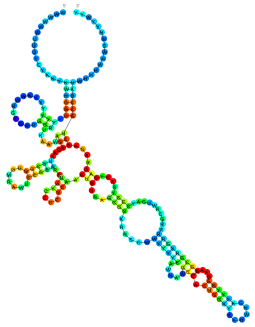

Figure 2.9: Cobalamin riboswitch family consensus secondary structure

Table 2.2: Bigram representation of figure 2.9

Bigram	Occurrence rate
dangling end – stem	1
stem – internal loop	2
internal loop – stem	2
stem – hairpin loop	4
stem – multi loop3	1
multi loop3 - stem	2
stem – multi loop4	1
multi loop4 - stem	3
stem - bulge	1
bulge - stem	1

Defining family-specific rules for producing bigrams

Is defining the set of all possible pairs of structural features which might be found in the sample secondary structures. For coarse shape learning, the following explicit rules build this set:

- Any type of loop is followed by a stem.
- Any type of loop except for hairpin loop follows a stem.

There are no explicit rules for dangling ends. They might or might not appear in a combination and for the current experiment, they do not have an impact on the calculation of objective function. The following is the list of valid bigrams for RF00174 family in python code. This is the basic grammar for analyzing sample RNA structures and the main reference for producing new ones. The vector representation of the consensus structure is generated based on this grammar.

Listing 2.1: basic grammar and consunsus bigram vector of RF00174

```
biGrams = [('dangling_end', 'stem') , ('multiloop3', 'stem') ,
           ('stem','multiloop3') , ('multiloop4' , 'stem') ,
           ('stem', 'multiloop4') , ('bulge', 'stem') ,
           ('stem','bulge') , ('internal_loop', 'stem') ,
           ('stem','internal_loop') , ('stem' , 'hairpinloop') ]

consensusBigram = (1, 2, 1, 3, 1, 1, 1, 2, 2, 4)
```

2.2.8 Reinforcement learning setup

In this section we explain the setup of the learning system we have used for the first set of experiments. We also talk about the important implementation facets of the system in python language.

The environment

Our simulation environment is called *Block Inverse Folding*. Block Inverse Folding combines the structural features resulted from decomposition of the set of secondary structures which belong to a Rfam family and outputs the new combinations. At each time step, one structural feature is selected from feature corpus and is added to the combination graph. Q-learning is used as the learning method, so no model of the environment is needed.The only environment dynamic used is the objective function.
Block Inverse Folding is an example of a domain-specific environment which we will refer to as *plant*. The modular structure of the code makes it possible to switch plants and solve another reinforcement learning problem.

The agent

The agent's functionality is split and implemented in two parts, action selection and learning, and interacting with the environment. In this system, there is no explicit entity as an agent, rather these functionalities are integrated in the plant and another class called the *controller*. Controller also contains the implementation of the learning algorithm used (here Q-learning). Here call it *QController*. It operates in two different modes:

- **Training** Here the learning is turned on, so the controller combines exploration with exploitation and as the result, outputs a final policy in the form of a Q-table.

- **Synthesis** Here no learning takes place. The controller loads a policy and starts to synthesize structures according to the utilized policy.

Episodic learning task

Learning is done in episodes. Each episode starts with an empty structure graph and is carried through by selecting a structural feature from the fea-

ture corpus and adding it to the state graph at each time step. At the end of each episode a final reward is calculated and given to the final state-action pair. An episode ends when:

- there are no open ends in the state graph or
- there are no more possible actions for the current structure or
- the number of structure bigrams is greater than the number of consensus structure bigram

Defining the state space

State is a part of the plant and contains several components:

- State graph: the graph which is the result of combination operation at each time step
- State vector: an ordered vector of integers as place-holders for state graph bigrams

Technically speaking, the state graph is not directly taking part in the learning operations. It is actually an aspect of the simulation environment which influences the state vector, learning trajectories, and the objective function. The state vector is the formal state representation which shapes the actual state space S.

Defining the action space

The action sequence in the learning environment is the sequence of structural features selected at each time step during a trial episode. Since the state vector is a bigram representation of the resulting graph, actions are also defined on a bigram basis. An action of the form *"X – Y"* indicates that a structural component of type *"X"* is already present in the state graph and has some open ends. The system then retrieves a structural component of type *"Y"* from the pool and attaches it to one of the open ends of X in the graph.
The system is designed to automatically build up the action set based on the structural features defined in the setup. Since some actions are impossible

to do, the program is designed to filter them in the action space definition phase.

Observations

Observations are all the information that the environment feeds back to the actor-critic system. Normally they are a full or partial observation of the state. In our setting, the state vector is the feed back to the controller.

The objective function

In our first set of experiments and for the sake of simplicity, we have defined an objective in the form of the bigram representation of the consensus secondary structure of the given Rfam family and we call it consensus bigram representation. This representation is formatted exactly as the state vector of the plant.
The reward function is then defined based on the differences between the state vector and consensus vector. To conduct the learning agent towards more promising structures, we designed the following reward calculation scheme:

- Bigrams which are present in both the consensus bigram representation and the state vector get a positive reward.
- Bigrams which exist in the state vector but do not have a counterpart in consensus bigram representation receive a negative reward.
- The final reward is a waited combination of these two.

Listing below shows the code of the reward function.

Listing 2.2: Reward function used in coarse shape learning

```
1 def get_reward(**opts):
2   consensus_dist=consensus_distance(flip=False,**opts)
3   consensus_diff=consensus_distance(flip=True,**opts)
4   plant=opts['plant']
5   default_reward=opts['default_reward']
6
```

```
7   if plant.is_terminal(opts):
8     reward=(5/exp(sum(consensus_dist[1:])))
9                   -(.2*exp(.2*sum(consensus_diff[1:])))
10    return reward
11  else:
12    return default_reward
```

Reinforcement learning setup summary

In short, our first reinforcement learning setup has the following specifications:

- Q-learning
- Episodic
- Discrete finite state space
- Discrete finite action space
- ϵ-greedy action selection method
- Delayed reward

2.2.9 System architecture

The system is called *RLRNA* and has a modular architecture both in design and implementation. It consists of two subsystems, the first one produces the feature corpus and maintains all the related operations. The second one is a reinforcement learning framework which carries out the actual learning. Here we talk about them briefly.

RNA decomposition subsystem

The RNA decomposition subsystem is a set of python packages which contain functions for carrying out all the operations related to decomposition, combination, visualization and storing of RNA structures. The main packages in this subsystem are:

- *RNA decomposition* for decomposition and combination operations
- *Graph* for special graph and graph statistics operations
- *workSuit* for general operations such as special file input/output and general probabilistic operations

Reinforcement learning framework

Is an Object Oriented framework in which, all the important learning elements are classes. Each class is written in an individual package and has generic interface for interacting with other classes. Therefore, it is possible to replace any part of the learning system in order to change its behavior. The main classes in the system are:

- *plant* is the actual simulation environment, here it is called Block Inverse Folding.
- *controller* is the actor-critic system which is responsible for selecting actions and learning. Here we use a QController which is performing Q-learning.
- *reward* is responsible for calculating the rewards.
- *Grammar* is domain-specific and contains the feature corpus database and provides other systems with data access operation.
- *Params* is a public container in order to provide an information-sharing mechanism between other objects.
- *main* is the kernel which initializes and orchestrates all other modules and runs the actual experiments.

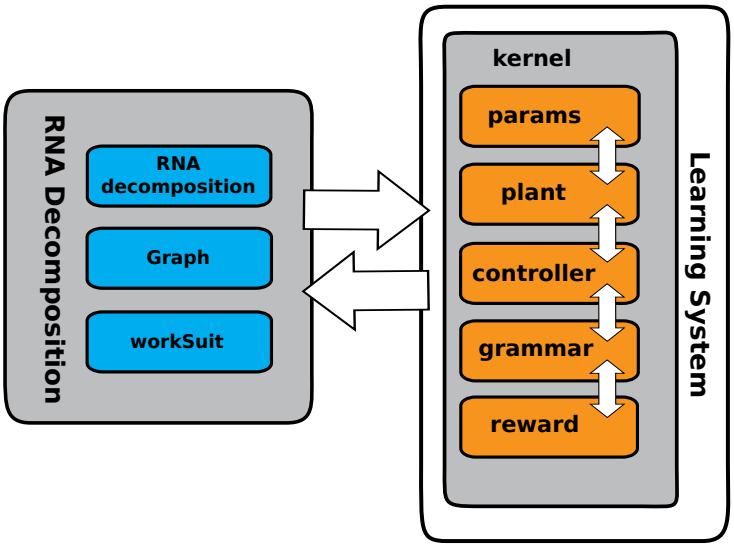

Figure 2.10: RLRNA system architecture

2.3 Refining RNA structure learning

The aim of this part of experiments is to train a model which can build combinations of secondary structure features with the same functionality as the sample RNA family. The measure which we use here is different from the previous section. There, we used the similarity between the produced combination and the consensus secondary structure of a family. In successful cases, this means that our synthesized product is an end-to-end sequence of nucleotides which are connected via backbone bonds.

Though when we use a folding algorithm to predict the secondary structure of the produced sequence, the same secondary structure is not predicted. This result is expected, since our features, and hence our states, do not contain enough information for the learning system to guide it towards an expected functionality.

In this chapter we introduce a mechanism for extracting refined information from the structural features. We integrate this mechanism into the old system and start a new set of learning experiments with a refined state space and action space definition. The measure here is a mixture of the old measure and the covariance score of the synthesized sequence.

2.3.1 Refined structural features and grammar

In the previous experiment we used 8 basic secondary structure feature names to generate a feature corpus. These feature names carry no further information about the actual subgraph except for its type. So it is necessary to add to the amount of information that each subgraph carries. This results to introducing new structural features which are built upon the basic structural features.

Forming the new feature space

The grammar module is still responsible for extracting structural features. Once the family corpus is built, the system can calculate statistical measures. In these series of experiments we add three groups of information:

- Relative GC-content
- Relative size
- Relative location in the original sequence

First we explain what we mean by relative location in the original sequence. If we divide an RNA sequence into k sections of size m, we can define the relative location of a structural feature in the graph as follows: A subset of the set {1, 2, . . . ,k } which presents all the sections to which the structural feature partially or completely belongs .
We have designed the system in a way that any number of other feature extraction functions could be added to the existing collection. In order to form the feature database, the system goes through the following steps:

- **Calculating statistics** This step is carried out by a special class called *FeatureExtractor*. This class takes the family corpus and calculates size and GC-content statistics in the form of percentiles. The number of percentiles is variant and is defined by a parameter. This information is used in the next step. Relative location statistics are derived differently, the FeatureExtractor makes a mapping between individual RNA sequences and their respective section dividers. This mapping is then used in the classification phase.

- **Extracting classification data** We use a discrete classifier with one feature which is a number. This classifier is ordinal and needs classification boundaries in the form of a ordered list of numbers. Given a number which is the GC-content or size, the classifier checks the boundaries between which the number falls and outputs a label. At this step the percentiles from the last step are used to define classification boundaries.

- **Producing new structural features** Here the system classifies all the structural features in the family corpus based on their GC-content and size and relative location and creates new features which their names are a combination of classification results and the basic feature type.

Concurrent to these operations, a new feature corpus is formed which contains all the new feature types as keys and all the subgraphs which are of the same type as a value list. Figure 2.11 shows chapter names in a refined feature corpus.

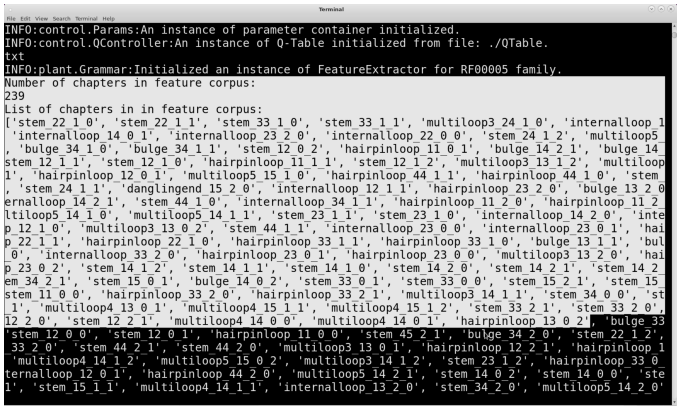

Figure 2.11: Refined feature corpus

In the following, we present the actual code snippets which are responsible for carrying out the operations we already described.

Listing 2.3: calculating structural feature size classification boundaries

```
1
2 def size_percentiles(q = None,iterable = None):
3   iterable = [nx.number_of_nodes(graph) for graph in iterable]
4   filtered = []
5   for item in iterable:
6           if not(item in filtered):
7                   filtered.append(item)
8   plist = create_percentile_list(q)
9   result = np.percentile(filtered,plist)
10  return result
11
12 def size_discriminants(q = 3,iterable = None):
13  p = size_percentiles(q=q,iterable=iterable)
14  return discriminants_from_percentiles(percentiles=p)
```

Listing 2.4: calculating sequence section boundaries

```
1 def section_discriminants( k = 15,graph = None ):
2   nodes =  nx.number_of_nodes(graph)
3   partition_size = int( nodes/k )
4   discriminants = [i for i in range(0 ,nodes ,k)]
5   return discriminants
```

Listing 2.5: finding the relative location of a given structural feature

```
1 def section_discriminants( k = 15,graph = None ):
2   nodes =  nx.number_of_nodes(graph)
3   partition_size = int( nodes/k )
4   discriminants = [i for i in range(0 ,nodes ,k)]
5   return discriminants
```

Automatic extraction of RNA sequence bigram representation

This functionality is added to the system to derive the bigram representation of a single given sequence. Each bigram consists of pairs of new structural features. This can be used to collect statistics and to infer and form a certain grammar. For now and in this series of experiments, we use this function to form a prototype which the learning uses as a part of its objective function.

2.3.2 Reinforcement learning setup

Here we use the same reinforcement learning system specifications. The architecture of the implementation allows changes to the plant and reward modules while the controller and kernel stay untouched. For operations that we introduce to the system, a new module is added.

Automatic action-set extraction

Actions have the same format as in coarse shape learning. They are in the form of pairs with the first element being an existing structural feature in the state graph and the second one being a structural feature which will get connected to that. An automatic action extraction procedure has been designed. Given the set of structural features, generates the entire action space as a list bigrams. Figure 2.12 shows a part of the action set extracted from RF00005 family.

Figure 2.12: Refined action space

Filtering the action-set

There are several groups of connection operations which are impossible to carry out in practice. All the pairs whose first element is a 'hairpin loop' are an example. In order to filter out 'impossible' operations, we set several rules which are extracted manually. Actions are explicitly generated by these rules. These rules are as follows :

- A 'dangling end' is always followed by a 'stem'.
- A 'hairpin' loop always follows a 'stem'.
- All other loop types can either follow or be followed by a 'stem'.

Fine-grained state representation

The state space of our learning problem has the following components:

- A state graph
- A state sparse vector

Considering the huge size of the action space, we switched to a sparse representation of the state graph. In the following , we present the code for state vector operations.

Listing 2.6: updating the state sparse vector given a set of new bigrams

```
def update_sparse_vector(global_model=None,sparse_vector=None,
                         new_biGrams= None,value=1,increase=True):
  if not(increase):
    value = -value
  for index, item in enumerate(global_model):
    if item in new_biGrams:
      if not(index in sparse_vector.keys()):
        sparse_vector.update({index:0})
        sparse_vector[index] = sparse_vector[index] + value
  return sparse_vector
```

Listing 2.7: converting state sparse vector to tuple

```
def sparse_vector_to_tuple(sparse_vector=None):
  vector_to_list = []
  for key in sorted(sparse_vector.keys()):
    vector_to_list.append((key,sparse_vector[key]))
  return tuple(vector_to_list)
```

Listing 2.8: converting state tuple to sparse vector

```
def tuple_to_sparse_vector(sparse_tuple = None):
  sparse_vector = {}
  for index, item in sparse_tuple:
    if not(index in sparse_vector.keys()):
      sparse_vector.update({index:0})
      sparse_vector[index] = sparse_vector[index] + item
  return sparse_vector
```

Refined reward function

The reward scheme suggested here is different from previous design. Since learning is concentrated in two main objectives rather than one, the reward should reflect them in a way which persuades the agent to achieve these objectives without being confused:

- learning the structure (coarse shape)
- learning the arrangement of nucleotides in a structure (functionality)

In the beginning of learning, the agent should concentrate on constructing structures with a specific shape. It learns to choose proper combinations of structural features. As the learning goes on, it narrows down its choices to those combinations which a specific arrangement of nucleotides which guarantee to pass in a certain scoring system. The suggested scoring is the covariance score. A program like Infernal [19] can be used to calculate this score. The following pseudo code shows how these two objectives could be put together.

Listing 2.9: Reward function used in refined strucutre learning

```
def get_reward( **opts ):

  #calculate a time dependent discount facotr gamma
  gamma = 1 / t
  #calculate the coarse shape reward (other reward functions could be used)
  coarse_reward=(a/exp(sum(consensus_dist[1:])))
                  -(b*exp(c*sum(consensus_diff[1:])))

  #calculate the covariance score of the sequence
  cov_score=score from infernal
  #calculate the actual reward
  reward = gamma*coarse_reward + (1-gamma)*cov_score
  return reward
```

2.3.3 System architecture

The architecture of the system does not vary from the previous version. The only difference is the addition of FeatureExtrator which is shown in figure 2.13.

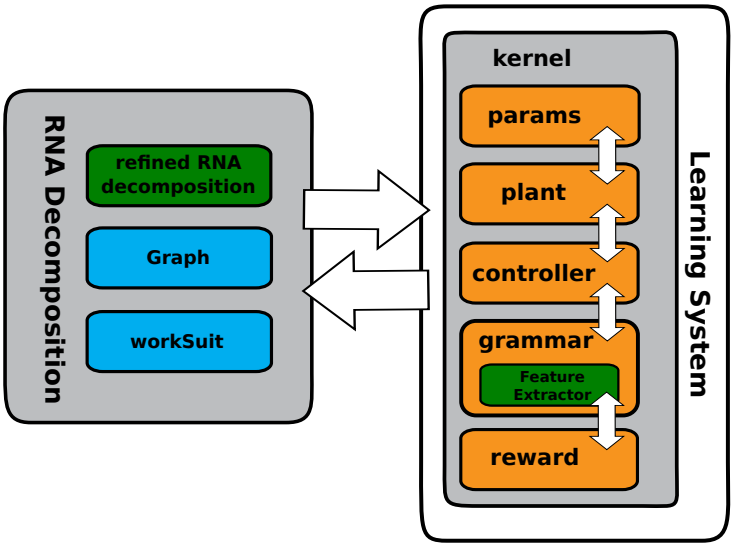

Figure 2.13: Refined RLRNA system architecture

Chapter 3

Experimental setup

In this chapter, actual experiments, their setup, and the results are discussed. Three groups of experiments of coarse shape learning are conducted on three different families of non-coding RNAs: RF00005, RF01685, and RF00174.

3.1 System parameters

All experiments use parameter settings which specify how the system learns and produces results. There are three main parameter groups, the first two groups are used in coarse shape learning and refined structure learning. The third group is just used in refined structure learning.

3.1.1 Learning parameters

These parameters control the behavior of the learning system.

episodes

number of episodes in each experiment run

epsilon

probability of performing exploration during learning ϵ

alpha

the learning rate factor α

default_reward

the first-visit reward of state-action pairs

initial_q_value

the value for initializing Q-table items

training

boolean option for switching between learning and synthesizing modes

3.1.2 General system parameters

These parameters specify the way the system outputs results or uses previously produced results.

save

boolean option for saving the model

qt_save_path

operating system path in which the trained model is saved

qt_save_frequency

frequency in number of episodes with which the model is saved on disk

load

boolean option for loading an existing model

visualize

boolean option for showing the state graph at the end of each episode

3.1.3 Feature extraction parameters

These parameters control the level of granularity for defining new structural features. There are three parameters in this group:

size_partition

Depicts the number of dividing boundaries between size categories for classifying a structural feature. This can be interpreted as : relatively small, medium, and large structural feature size.

gc_partition

Depicts the number of dividing boundaries between GC-content categories for classifying a structural feature.

section_size

Depicts the size of sections based on sequence length. By sequence length we mean the number of nucleotides in the sequence.

3.2 Experiments

Experiments are run in RLRNA simulation environment. Each experiment consists of a batch of episodes of online Q-learning. The end result of each experiment is a Q-table in case the system is learning or a list of synthesized structures in networkx graph format in case the system is in synthesis mode.

3.2.1 Parameter setup

Learning and general parameter settings for this experiment are listed below:

Table 3.1: Learning parameter setting for coarse shape learning

Parameter	Value
epsilon	0.5
alpha	1.0
gamma	1.0
default_reward	0.0
initial _q _value	0.0
training	True
episodes	10000

Table 3.2: Training mode parameter setup

Parameter	Value
visualize	False
save	True

Table 3.3: Synthesis mode parameter setup

Parameter	Value
visualize	True
load	True

3.2.2 Experiment1: tRNA

tRNA family (RF00005) has 954 representative (seed) sequences. This family has a symmetric consensus secondary structure. The system successfully learned the coarse shape after 80000 episodes, though it was not resemble the only bulge which is seen in the consensus structure.

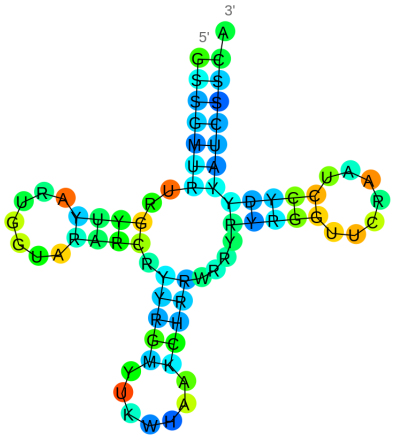

Figure 3.1: tRNA consensus secondary structure[1].

Learning the above structure converged in 70000 episodes. In the following, a report of the learning process, some sample structures produced by the learned model, and the learning curve are demonstrated.

Table 3.4: tRNA coarse shape learning report

correct structures (%)	Episodes	Time elapsed
5	20000	400 seconds
24	30000	600 seconds
34	50000	1000 seconds
100	70000	1400 seconds

[1] Available under `http://rfam.xfam.org/`, seen on 24.08.2015.

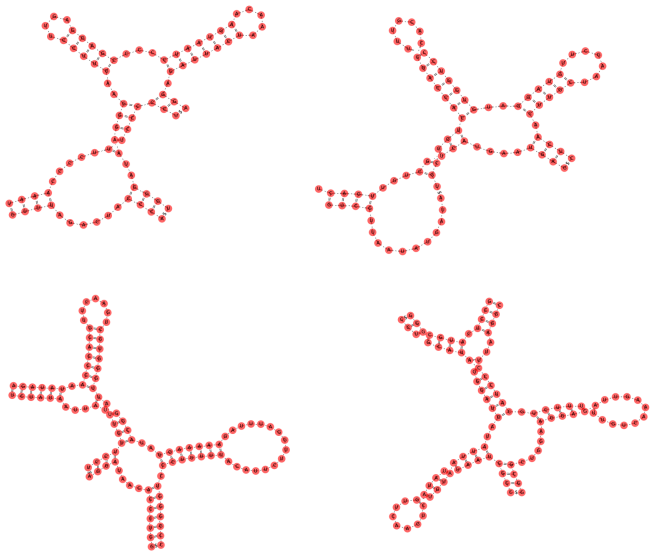

Figure 3.2: Sample structures after 10000 learning episodes

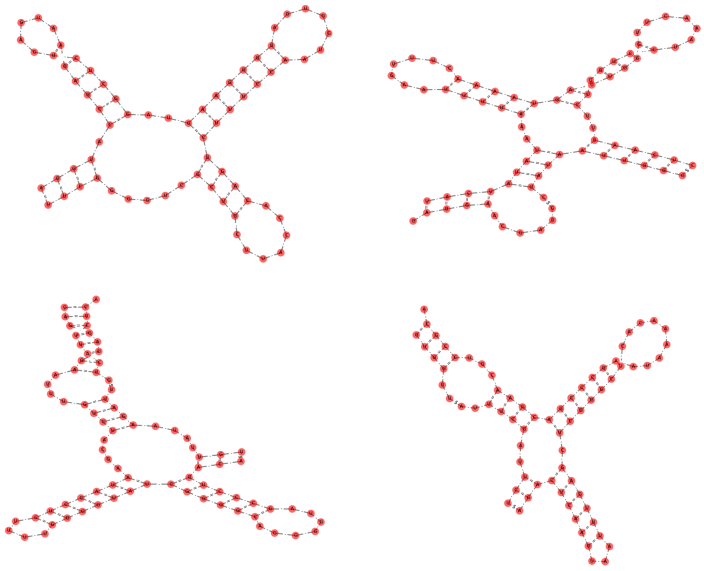

Figure 3.3: Sample structures after 30000 learning episodes

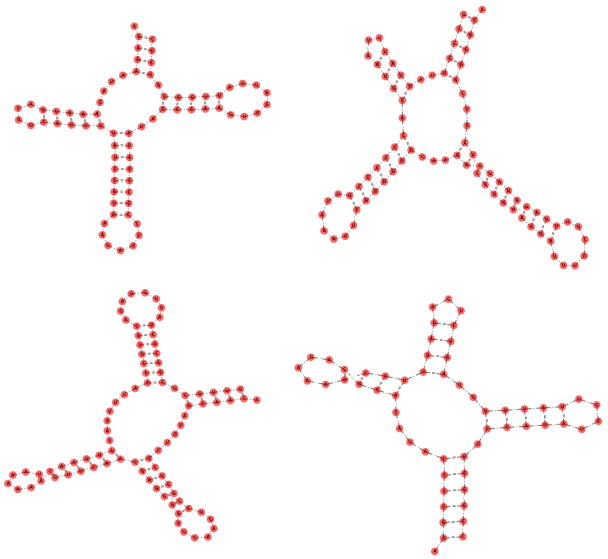

Figure 3.4: Sample structures after 70000 learning episodes

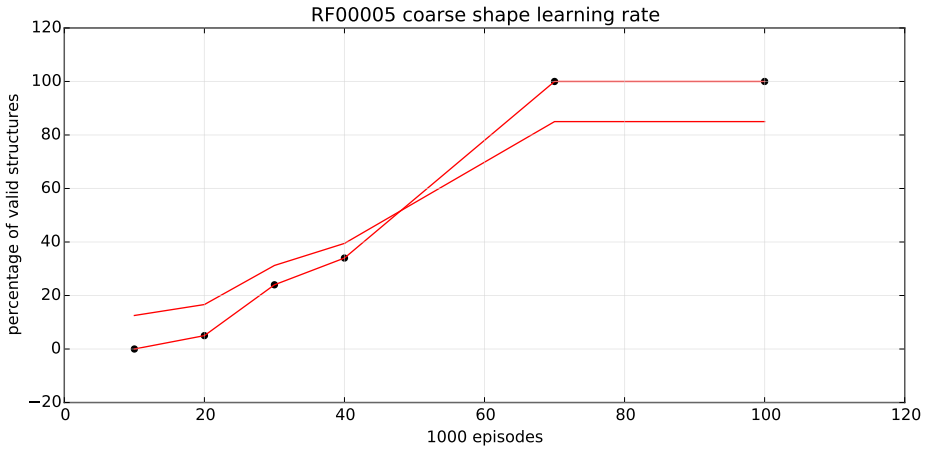

Figure 3.5: tRNA coarse shape learning curve

3.2.3 Experiment2: 6S-Flavo

6S-Flavo (RF01685)family has 82 representative members[2]. The consensus secondary structure of this family has a relatively simple shape.

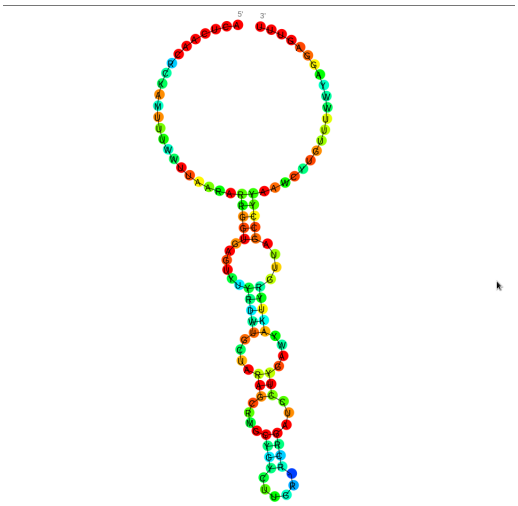

Figure 3.6: 6S-Flavo consensus secondary structure

Learning the above structure converged in 80000 episodes. In the following, a report of the learning process, some sample structures produced by the learned model, and the learning curve are demonstrated.

Table 3.5: 6S-Flavo coarse shape learning report

correct structures (%)	Episodes	Time elapsed
29	20000	180 seconds
35	30000	300 seconds
56	50000	500 seconds
100	80000	830 seconds

[2]Available under `http://rfam.xfam.org/`, seen on 24.08.2015.

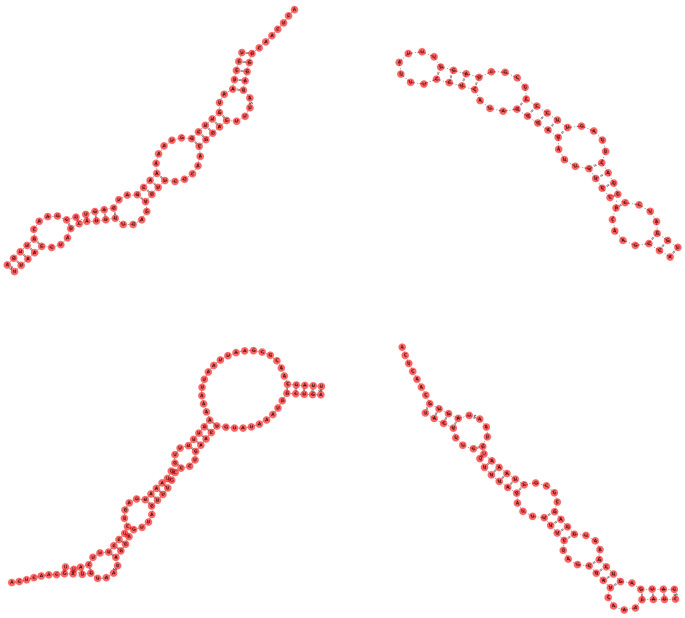

Figure 3.7: Sample structures after 20000 learning episodes

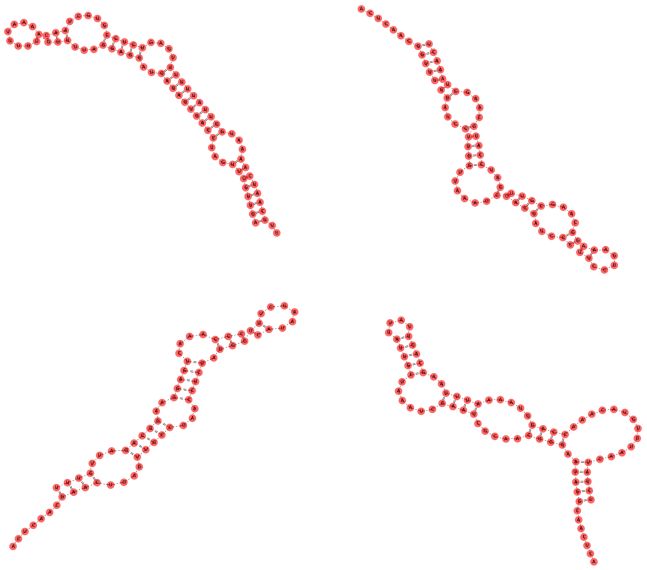

Figure 3.8: Sample structures after 80000 learning episodes

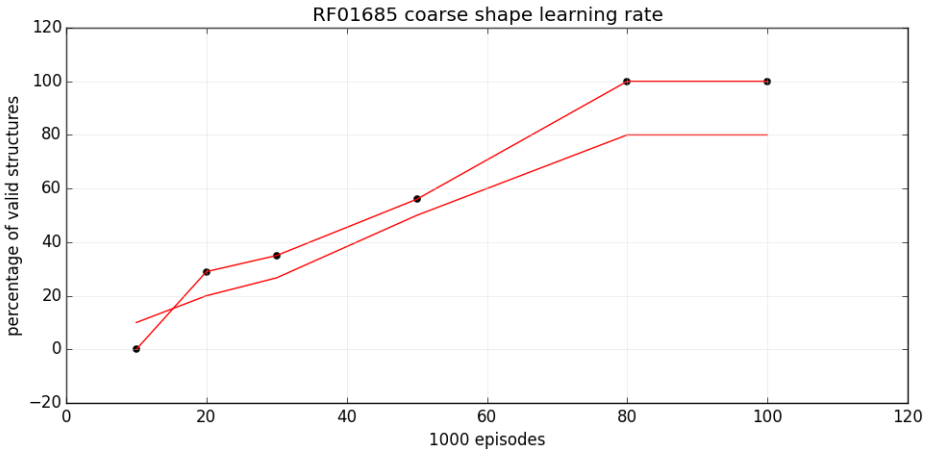

Figure 3.9: RF01685 coarse shape learning curve

3.2.4 Experiment3: Cobalamin riboswitch

Cobalamin riboswitch (RF00174) with 430 representative members has a more complicated, non-symmetric shape. The system failed to converge in this case and with the current parameter settings.

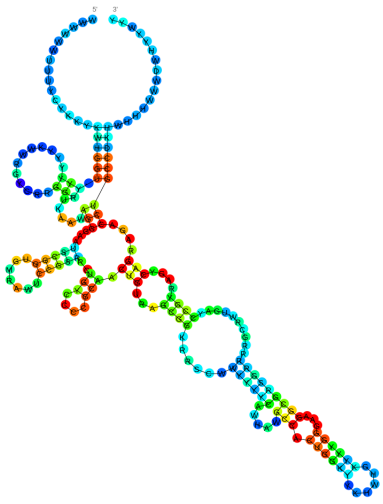

Figure 3.10: Cobalamin riboswitch consensus secondary structure

Chapter 4

Discussion and future work

Experiments in the previous chapter showed that the system is capable of learning simple or symmetric structures with a default parameter setting and in linear time. More complicated structures need to be studied further. Possible courses of action in this regard are:

- Running the experiments with different groups of parameter settings
- Designing and testing other reward functions
- Using higher orders of connectivity for structure analysis and synthesis
- Testing and completing the refined learning system

In fact the refined learning scheme can implicitly use a higher order of neighborhood distance in a simplified way, since it encodes the locale information of substructures as one of their attributes. There are also some suggested refinements to RNA decomposition system. The recombination operations are designed to connect substructures by taking their sequence direction into consideration, though there is no check in the flipping of a stem. Here a mechanism is suggested to also tag the stems with a primary and secondary end to solve this problem. It is also highly suggested that RNA decomposition system is tested upon many different Rfam groups and outside of the learning context.

To lead the current work towards a robust production system, there are several notes which are discussed here:

- The refined learning scheme has passed the design stage and also the first implementation phase. Future results from refined shape learning experiments should make a big step to benchmark and improve the system.
- Once the systems are stable, parameter optimization should be applied to them.
- Grammar inference rules are user-defined, the next implementation could embrace a probabilistic inference system for producing grammar.
- The current learning system uses a Q-table which could be replaced by a function approximation mechanism in future versions.

Future awaits!

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 sinngemäß 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.

Ort, Datum	Unterschrift

Bibliography

[1] Mirela Andronescu, Anthony P. Fejes, Frank Hutter, Holger H. Hoos, and Anne Condon. A new algorithm for {RNA} secondary structure design. *Journal of Molecular Biology*, 336(3):607 – 624, 2004.

[2] Guillaume Blin, Guillaume Fertin, Irena Rusu, and Christine Sinoquet. Extending the hardness of rna secondary structure comparison. In *In intErnational Symposium on Combinatorics, Algorithms, Probabilistic and Experimental methodologies (ESCAPE), LNCS*. Springer, 2007.

[3] Anke Busch and Rolf Backofen. Info-rna—a fast approach to inverse rna folding. *Bioinformatics*, 22(15):1823–1831, 2006.

[4] William B. Cavnar and John M. Trenkle. N-grambased text categorization. In *In Proc. of SDAIR-94, 3rd Annual Symposium on Document Analysis and Information Retrieval*, pages 161–175, 1994.

[5] Fabrizio Costa and Kurt De Grave. Fast neighborhood subgraph pairwise distance kernel. In *Proceedings of the 26th International Conference on Machine Learning,*, 2010.

[6] Sean R. Eddy and Richard Durbin. Rna sequence analysis using covariance models. *Nucleic Acids Research*, 22(11):2079–2088, 1994.

[7] Ali Esmaili-Taheri, Mohammad Ganjtabesh, and Morteza Mohammad-Noori. Evolutionary solution for the rna design problem. *Bioinformatics*, 30(9):1250–1258, 2014.

[8] European Commission. Directorate General for Research. *Synthetic Biology: Applying Engineering to Biology : Report of a NEST High-Level Expert Group*. Community research: Project Report. Office for Official Publications of the European Communities, 2005.

[9] M. GANJTABESH. *RNA Structure Prediction: The Inverse Problem*. Department of Computer Science, University of Tehran,and School of Biological Sciences, IPM.

[10] Juan Antonio Garcia-Martin, Ivan Dotu, and Peter Clote. Rnaifold 2.0: a web server and software to design custom and rfam-based rna molecules. *Nucleic Acids Research*, 43(W1):W513–W521, 2015.

[11] I.L. Hofacker, W. Fontana, P.F. Stadler, L.S. Bonhoeffer, M. Tacker, and P. Schuster. Fast folding and comparison of rna secondary structures. *Monatshefte für Chemie / Chemical Monthly*, 125(2):167–188, 1994.

[12] Christian Höner zu Siederdissen, Stefan Hammer, Ingrid Abfalter, Ivo L. Hofacker, Christoph Flamm, and Peter F. Stadler. Computational design of rnas with complex energy landscapes. *Biopolymers*, 99(12):1124–1136, 2013.

[13] Robert Kleinkauf, Martin Mann, and Rolf Backofen. antarna: ant colony-based rna sequence design. *Bioinformatics*, 2015.

[14] Pedro Larrañaga, Borja Calvo, Roberto Santana, Concha Bielza, Josu Galdiano, Iñaki Inza, José A Lozano, Rubén Armañanzas, Guzmán Santafé, Aritz Pérez, et al. Machine learning in bioinformatics. *Briefings in bioinformatics*, 7(1):86–112, 2006.

[15] Rune B Lyngsø, James Wj Anderson, Elena Sizikova, Amarendra Badugu, Tomas Hyl, and Jotun Hein. Research article open access frnakenstein: multiple target inverse rna folding.

[16] John S. Mattick and Igor V. Makunin. Non-coding rna. *Human Molecular Genetics*, 15(suppl 1):R17–R29, 2006.

[17] Thomas M. Mitchell. *Machine Learning*. McGraw-Hill, Inc., New York, NY, USA, 1 edition, 1997.

[18] Eric P. Nawrocki, Sarah W. Burge, Alex Bateman, Jennifer Daub, Ruth Y. Eberhardt, Sean R. Eddy, Evan W. Floden, Paul P. Gardner, Thomas A. Jones, John Tate, and Robert D. Finn. Rfam 12.0: updates to the rna families database. *Nucleic Acids Research*, 2014.

[19] Eric P. Nawrocki, Diana L. Kolbe, and Sean R. Eddy. Infernal 1.0: inference of rna alignments. *Bioinformatics*, 25(10):1335–1337, 2009.

[20] Vladimir Reinharz, Yann Ponty, and Jérôme Waldispühl. A weighted sampling algorithm for the design of rna sequences with targeted secondary structure and nucleotide distribution. *Bioinformatics*, 29(13):i308–i315, 2013.

[21] Stuart J. Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Pearson Education, 2 edition, 2003.

[22] Michael Schnall-Levin, Leonid Chindelevitch, and Bonnie Berger. Inverting the viterbi algorithm: An abstract framework for structure design. In *Proceedings of the 25th International Conference on Machine Learning*, ICML '08, pages 904–911, New York, NY, USA, 2008. ACM.

[23] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*, volume 28. MIT press, 1998.

[24] Akito Taneda. MODENA: a multi-objective RNA inverse folding. *Advances and applications in bioinformatics and chemistry : AABC*, 4:1–12, 2011.

[25] Julie D. Thompson, Frédéric Plewniak, and Olivier Poch. A comprehensive comparison of multiple sequence alignment programs. *Nucleic Acids Research*, 27(13):2682–2690, 1999.