Master's Thesis

Modular and Linear-Time Structural Clustering of RNA Sequences using the Galaxy Framework

Eteri Sokhoyan

April 2017

Albert-Ludwigs-Universität Freiburg im Breisgau
Technische Fakultät
Institut für Informatik

First Examiner	Prof. Dr. Rolf Backofen
Second Examiner	Prof. Dr. Wolfgang Hess
Supervisors	Milad Miladi, Dr. Bjoern Gruening

Declaration

Zusammenfassung

Die Erforschung von non-coding RNA ist ein relativ neues Forschungsgebiet und die Funktion vieler non-coding RNAs ist bis zum heutigen Tage nicht bekannt. Das Clustering von Sequenzen ist die allgemein akzeptierte Methode zur funktionellen Annotation. Es ist bekannt, dass die Sekundärstruktur von ncRNAs besser erhalten bleibt als die Sequenz, weswegen das Clustering basierend auf dieser Vorteile gegenüber dem rein sequenzbasiertem Clustering hat. Die GraphClust Pipeline, veröffentlicht im Jahr 2012, ist eines der besten Verfahren zum Clustern von alignment-freien Sekundärstrukturen. Seine lineare Zeitkomplexität ermöglicht es mit bis zu mehreren hunderttausend Sequenzen zu arbeiten.

Im Rahmen dieser Arbeit stellen wir eine neue und verbesserte Version von GraphClust vor. Die Integration dieser in das Galaxy Framework ermöglicht nicht nur eine einfache Nutzung, sondern ist auch flexibel und modular. Benutzern wird hierdurch ermöglicht das Tool mittels ihres Web-Browsers, unabhängig von ihrem Betriebssystem zu nutzen. Des Weiteren ermöglicht dies eine Erweiterung und den Austausch von Teilen der Pipeline, ohne tief in den Programmcode eingreifen zu müssen.

Wir zeigen anhand einer vergleichenden Analyse, dass unsere Methode bessere Ergebnisse als die originale Pipeline erziehlt. Außerdem präsentieren wir beispielhaft Ergebnisse unserer Methode angewandt auf einen Metatranscriptome Datensatz, welcher 100000 Sequenzen enthält.

Abstract

Non-coding RNA research is a relatively new field and there are still many non-coding RNAs with unkown function. Clustering sequences is an accepted way for functional annotation of ncRNAs. As it is known, for ncRNAs the secondary structure is better preserved than the sequence, hence, clustering based on the secondary structure has an advantages over that on sequences. The GraphClust pipeline, introduced in 2012, is by far one of the best alignment-free secondary structure clustering approaches. Its linear time complexity enables users to work with hundreds of thousands of sequences.

We present an upgraded and improved version of GraphClust. Integration of the pipeline in Galaxy framework makes it not only easy to use but also more flexible and modular. Users can deploy the pipeline via web browser, independent from their working platform. Furthermore, it is possible to extend or exchange parts of the pipeline without touching the source code.

Moreover, we present the results of a comparative analysis of our approach and show that it is performing better than the original pipeline. Furthermore, we present results of a use case on a metatranscriptome dataset with 100000 sequences.

Contents

1 Introduction

Biology is a field of life sciences, which aims to study all living entities e.g humans, animals and plants. It evolved over centuries and led to the foundation of many specialized subfields. One of these is bioinformatics. Bioinformatics can be considered a bridge between life sciences and computer science. Due to the infeasibility of humans to handle big amounts of data, biology "merged" with computer science and resulted in bioinformatics. Bioinformatics supports life science by providing automated tools for solving life science problems. There are already many tools and methods that help scientists to answer biological questions. However, with fast developing computational resources, new discoveries are made in science, with a demand for novel techniques. For example, the discovery of non-coding RNAs opened a whole new field of a research. New discoveries are quickly changing our understanding of genome complexity and raise further challenging questions. That is why the comparison, prediction and functional annotation of non-coding RNAs are now one of the major problems of modern RNA research. Clustering RNA sequences has become a generally accepted way for ncRNAs annotation.

In 2012 Steffen Heyne and Fabrizio Costa presented an RNA clustering approach, called GraphClust, which is taking into account sequential, as well as the structural information. GraphClust [1] is a pipeline for alignment-free, structural clustering of local RNA secondary structures. It is based on graph kernels and local sensitive hashing techniques and aims to discover new ncRNA families, as well as to predict conserved structures. GraphClust is a complex pipeline consisting of different steps and using several third-party tools. It was proven to have linear run-time and thus is able to work with up to hundreds of thousands of sequences. However, there is room for improvement, especially with respect to modularity and accessibility.

In the scope of this work we explain the existing GraphClust pipeline in detail, discuss possible improvements and challenges and present our approach designed to solve these problems. Furthermore, we integrated it in the Galaxy framework [2], enabling users to use GraphClust via a web browser, without any restrictions concerning the platform they are working on. This also enables users to easily extend the pipeline with available Galaxy tools, in order to perform extra pre- or post-processing of the data, as well as to use the components of the pipeline independently.

Moreover, we make a comparative analysis of our approach with the original pipeline. We evaluate the qualities of the clusters obtained from both methods and show that our approach outperforms the original. Additionally, we demonstrate our approach on a use case with a metatranscriptome dataset dominated by cyanobacteria, and show that it can handle large-scale datasets.

2 Theoretical Background

In this chapter we give an overview of major biological, as well as technical concepts and techniques, that we used throughout this thesis.

2.1 Biological Overview

Deoxyribonucleic acid (DNA), ribonucleic acid (RNA) and proteins are three major biological macromolecules that are essential for various known forms of life. Fig. 2.1 shows relations of these three macromolecules, or so-called *central dogma* of molecular biology[3]. The central dogma of molecular biology describes the information flow in the genes from DNA to RNA to proteins. In the following sections we will give a brief overview of DNA and talk more detailed about RNA, as it is the basis of this work.

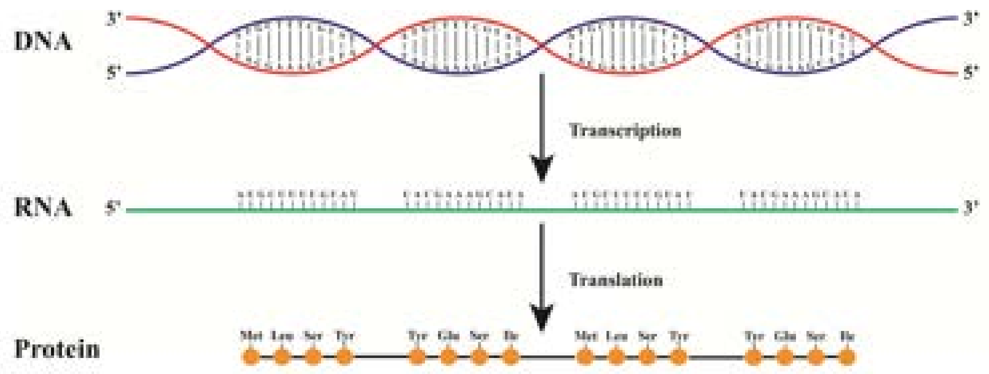


Figure 2.1: Information flow in biology. DNA contains instructions for making a protein, which are copied to RNA via transcription. Then RNA uses these instructions to synthesize a protein via translation [4].

2.1.1 DNA

DNA or Deoxyribonucleic acid molecule stores genetic information in living systems. This genetic information is crucial for a living system to live, develop and reproduce. A DNA molecule consists of a phosphate group, deoxyribose sugar and a nitrogen base. The nitrogen base can be one of the four nitrogen-containing nucleobases adenine (A), thymine (T), guanine (G) and cytosine (C). These bases can form pairs adenine - thymine and guanine - cytosine. These pairs are also known as complementary pairs [5]. In DNA the sugars and phosphates bind together to form two backbones. These backbones are attached to each other by the nucleobases. Because, as mentioned before, nucleotides pairing is done according to the rule A with T and G with C, thus one strand or backbone of DNA is the reverse complement of the other one [6]. The double stranded or double helix structure of the a DNA is shown in Fig. 2.2.

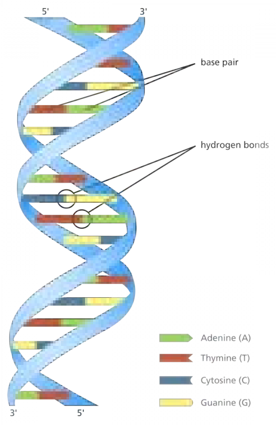


Figure 2.2: DNA double helix. The DNA molecule consists of two sugar-phosphate backbones or so-called strands. These strands held together by bonds between complementary bases A-T, G-C [7].

Transcription

As it was stated in the previous section, DNA contains genetic information that living system needs for existence. To be able to use this genetic information DNA is transcribed or summarized to a different molecule called RNA. This process is called *transcription*. So transcription is the first step of the process described by the central dogma : DNA to RNA (Fig. 2.1). It is the transfer of genetic instructions in DNA to messenger RNA (mRNA) (Sec. 2.1.4). During the transcription, a strand of mRNA is made in a way, that it is complementary to a strand of DNA [8]. Transcription consists of three steps *initiation*, *elongation* and *termination*. It starts with *initiation*. During this step the enzyme RNA polymerase binds to a specific region of a gene called promoter. The DNA untwists, such that the enzyme can “read” the bases in one of the DNA strands. The next step is *elongation*, where the addition of nucleotides to the mRNA strand occurs. RNA polymerase reads the DNA strand and builds the mRNA molecule, using complementary base pairs. During this process, an adenine (A) in the DNA binds to an uracil (U) in the RNA. The transcription is considered to be over when the RNA polymerase reaches a termination sequence in the gene. It means that mRNA is complete and detached from the DNA. This step is called *termination* [8].

Translation

While *transcription* is the process of information flow from DNA to RNA, *translation* is the process of making proteins : RNA to proteins (Fig. 2.1). After the termination step of transcription mRNA moves to a structure called ribosome. Ribosome is a complex molecule, with a small and a large subunits, consisting of rRNA (Sec. 2.1.4) and proteins. Just like transcription, translation process has three stages called *initiation*, *elongation* and *termination*. During the *initiation* step, the small subunit of the ribosome binds to the start of mRNA sequence. Next tRNA, which carries amino acid called methionine, binds to the start codon of mRNA. This start codon has the sequence *AUG* for all mRNA molecules. Then the large subunit of ribosome binds forming the complete initiation complex. The *elongation* stage one can imagine as an iterative process where in each round ribosome translates one codon of mRNA. This process is resulting a growing chain of amino acids linked by peptide bond. Translation process reaches the *termination* stage when the ribosome reaches a stop codon. The new protein is synthesized and the translation complex is disorganized [9].

2.1.2 RNA

RNA molecules, which are linear chains (or polymers) of ribonucleotides or just nucleotides, perform a number of critical functions. Many of these functions are related to protein synthesis. Some RNA molecules bring genetic information from a cell’s chromosomes to its ribosomes, where proteins are assembled. Others help ribosomes translate genetic information to assemble specific sequences of amino acids. Nucleotides, the building blocks of RNAs, consist of a ribose (a five-carbon sugar), a phosphate group and nitrogen-containing base. The nitrogen-containing base may be adenine (A), cytosine (C), guanine (G), or uracil (U). The nucleotides in RNA molecules are linked together to form chains. The link between two nucleotides is between a

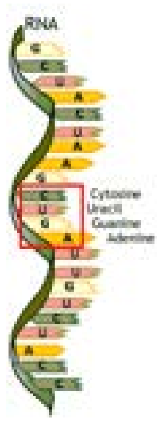


Figure 2.3: RNA strand. An RNA molecule consist of a ribose, which is a five-carbon sugar, a phosphate group and a nitrogen-containing base (A, C, G, U) [11].

phosphate group attached to the fifth (5' or "five prime") carbon of the sugar on one nucleotide and a hydroxyl group on the third (3' or "three prime") carbon of the sugar on the other. The link is called a 5'- 3' phosphodiester bond [10].

Whereas DNA is usually double-stranded (Fig. 2.2), with the bases on one strand pairing up with ones on the other, RNA usually exists as single chains of nucleotides (Fig. 2.3). The bases in RNA follow Watson-Crick base-pair rules: A and U can pair with each other, as can G and C. Wobble base pairs or non Watson-Crick base pairs are also possible, but these are considered as very rare events [5].

The RNA structure can be divided in three different levels, which are *primary*,*secondary* and *tertiary* structure. The *primary* structure can be denoted by lining up the bases from 5' to 3' in the order of the backbone. In other words the primary structure is a one dimensional representation of the RNA. The *secondary* structure of RNA is formed when bases form base-pairs. This can be considered as a two dimensional representation of RNA. And finally the three dimensional representation is called *tertiary* structure.

In the scope of this work, RNA *secondary* structure is the most relevant, that is why we will take a closer look into it.

2.1.3 RNA Secondary Structure

As outlined above, the RNA secondary structure is formed when two bases form a base-pair, i.e. two bases form hydrogen bonds with each other. The two bases that can form a base-pair are also called *complementary* bases. An

RNA secondary structure can be formally defined as shown in Def. 2.1.1.

Definition 2.1.1. *Let $S \in \{A, C, G, U\}*$ be an RNA sequence with length $n = |S|$. A Secondary structure of an RNA is a set of base pairs P, where $P \subseteq \{(i, j) \mid 1 \leq i < j \leq n,\ S_i \text{ and } S_j \text{ are complementary } \}$, such that P has degree of at most one, i.e. $\forall (i, j)(i', j') \in P : (i = i' \Leftrightarrow j = j')$ and $i \neq j$.*

The restriction on the degree of P is preventing the formation of *crossing* base-pairs, i.e. each base can form a base-pair with no more than one complementary base.

In literature one can find term *pseudoknots* for crossing base-pairs (Def. 2.1.2). The problem with pseudoknots is that computational cost for predicting structures containing them is expensive and that is why most RNA structure predicting algorithms do not consider them.

Definition 2.1.2. *Two base-pairs (i, j) and (i', j') are crossing iff $(i < i' < j < j')$ or $(i' < i < j' < j)$.*

If an RNA secondary structure does not contain any pseudoknots it is called a *nested* structure. RNA secondary structure consists of several elements: single-stranded bases enclosed by one or more base-pairs, stackings and loops. Fig. 2.4 is the graphical representation of RNA secondary structure elements and Def. 2.1.3 is the formalization of these elements.

Definition 2.1.3. *Let S be an RNA sequence and P be the structure for S.*

- ***Stacking*** *is formed by a base-pair $(i, j) \in P$ if $(i + 1, j - 1) \in P$*
- ***Hairpin loop*** *is formed by a base-pair $(i, j) \in P$ if $\forall i < i' \leq j' < j : (i', j') \notin P$*
- ***Internal loop*** *(i, j, i', j') is formed by base-pairs $(i, j) \in P$ and $(i', j') \in P$ if*
    - $i < i' < j' < j$
    - $(i' - i) + (j - j') > 2$ *(no stacking)*
    - *there is no base-pair (k, l) between (i, j) and (i', j')*
- ***Left (right) bulge*** *is a special case of internal loop when, $j = j' + 1$ $(i' = i + 1)$*
- *A **k-multiloop** consists of multiple base-pairs $(i_1, j_1)....(i_k, j_k) \in P$ with a closing base-pair (j_0, i_{k+1}) with following properties :*
    - $\forall 0 \leq l \leq k : (j_l < i_{l+1})$
    - *$\forall 0 \leq l,\ l' \leq k$ is true that there is no base-pair $(i', j') \in P$ with $i' \in [j_l....i_{l+1}]$ and $j' \in [j_{l'}....i_{l'+1}]$*

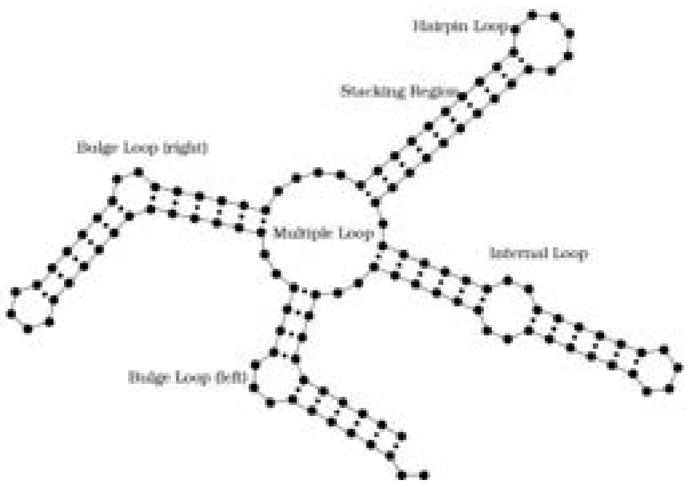


Figure 2.4: RNA secondary structure elements. Examples of typical elements found in an RNA secondary structure. Such elements are stacking regions (or helices), bulge loops, internal loops, hairpin loops and multiple loops [12].

2.1.4 Common Types of RNA

In this section we describe the most common types of RNA, which play an important role in the function of a cell and protein synthesis.

Messenger RNA

Messenger RNA or just mRNA accounts for just 5% of the total RNA in the cell. mRNA is the most heterogeneous of the 3 most common types of RNA in terms of both base sequence and size. It carries the genetic code copied from the DNA during transcription (Sec. 2.1.1) in the form of triplets of nucleotides called codons. Each codon specifies a particular amino acid, but one amino acid can be coded by many different codons. Although there are 64 possible codons or triplet bases in the genetic code, only 61 of them represent amino acids. The remaining three are stop codons [10].

Transfer RNA

Transfer RNA or tRNA is one of the smaller types of RNA, having about 75-95 nucleotides. tRNAs are essential components of translation, where their main function is the transfer of amino acids during protein synthesis. Therefore they are called transfer RNAs. Each tRNA molecule has an anticodon for the amino acid it carries. An anticodon is a sequence of 3 bases, and it is complementary to the codon for an amino acid. Each of the 20 amino acids has a specific tRNA that binds with it and transfers it to the growing polypeptide chain. tRNAs also act as adapters in the translation of the genetic sequence of mRNA into proteins. That is why they are also called adapter molecules [13].

Ribosomal RNA

Ribosomal RNAs or rRNAs are found in the ribosomes and account for 80% of the total RNA present in the cell. Both prokaryotic and eukaryotic ribosomes are composed of a large and a small subunits. In prokaryotes the large subunit is called $50S$ and small one $30S$, and in eukaryotes $60S$ and $40S$ respectively. Each of these subunits has its own rRNA molecules. Different rRNAs present in the ribosomes include small rRNAs and large rRNAs, which denote their presence in the small and large subunits of the ribosome.

rRNAs combine with proteins in the cytoplasm to form ribosomes, which act as the site of protein synthesis and have the enzymes needed for the process. These complex structures travel along the mRNA molecule during translation and facilitate the assembly of amino acids to form a polypeptide chain. They bind to tRNAs and other molecules that are crucial for protein synthesis.

In bacteria, the small and large rRNAs contain about 1500 and 3000 nucleotides, whereas in humans, they have about 1800 and 5000 nucleotides. However, the structure and function of ribosomes is largely similar across all species [14].

Non-coding RNA

As briefly mentioned earlier, not all RNAs are translated into proteins. These RNA molecules encoded by genes in genome are functional but non-translated into proteins called non-coding RNAs (ncRNAs). Alanine transfer RNA is the first discovered ncRNA [15], followed by many more afterwards. It is important to note that these ncRNAs have vital functions even though they are not translated into proteins. There are different groups of ncRNAs with various roles in cellular processes. In human genome the exact number of ncRNAs is unknown yet, however recent studies suggests the existence of thousands of ncRNAs. Recent studies in genomics have shown that in complex biological organisms, a major part of the genetic information is copied into ncRNAs [16]. This has brought a lot of attention to the study of structures and functions of different types of ncRNAs. In contrast to protein-coding RNAs, ncRNAs belong to a diverse array of classes with vastly different structures, functions, and evolutionary patterns. Moreover, ncRNAs can be divided into

RNA families, according to inherent functional, structural, or compositional similarities [17].

2.2 RNA Bioinformatics Background

2.2.1 Sequence Alignment

Sequence alignment is used in bioinformatics to help scientists to understand structural, functional and evolutionary relationships between sequences. The idea of a sequence alignment (Def. 2.2.1) is to transform one sequence to another by set of edit operations. Edit operations are *insertion*, *deletion* and *substitution*. One edit operation is done for a pair of residues (x, y), where x is an instance of sequence a and y of sequence b. *Insertion* is the procedure of inserting a gap in sequence a, *deletion* is inserting a gap in sequence b. *Substitution* is the process of aligning x with y when $x \neq y$. This is called a mismatch.

Definition 2.2.1. *Let $a, b \in \Sigma^*$ be 2 words and $\Sigma = \{A, C, G, U\}$.The alignment of (a, b) consists of 2 sequences $a^*, b^* \in (\Sigma \cup \{-\})$ such that:*

- $|a^*| = |b^*|$ *the length of 2 sequences is the same*
- $\nexists 1 \leq i \leq |a^*|$ *such that* $a_i^* = b_i^* = -$ *no column with only gaps is allowed*
- $a^* - \{-\} = a$ *and* $b^* - \{-\} = b$ *removing gaps from alignment sequences will result the original sequences*

To understand how strongly two sequences are related, or if related at all, a measurement of similarity is needed. If the similarity score is high, then sequences are similar and thus the distance score is low and vise versa. So to compute alignment based on similarity, the score should be maximized and in case of distance the score should be minimized. The most common distance metrics are the Euclidean distance, the Hamming distance and the Levenshtein distance. Each metric has its own definition on how to evaluate the distance between sequences. The Levenshtein distance, for example, counts the number of edit operations which are necessary to convert one sequence into another. The similarity measure of two sequences can be calculated through a scoring function. For that we need a cost function (Eq. (2.1)) which defines costs for match, mismatch and gap cases.

$$s(x, y) = \begin{cases} \alpha, & \text{if } x = y, \\ \beta, & \text{if } x \neq y, \\ \gamma, & \text{if } x = - \text{ or } y = - \end{cases} \tag{2.1}$$

A match should have a higher score than a mismatch, and a gap should be punished. Having such a scoring function, we can define a similarity score for an alignment (a, b) as shown in equation Eq. (2.2).

$$s(a, b) = \sum_{1 \leq i \leq |a|} s(a_i, b_i) \tag{2.2}$$

There are two types of alignment methods: global and local. Global alignment methods always align both sequences completely with each other. In contrast, local alignment methods try to find sections in both sequences that are highly similar [18].

2.2.2 RNA Secondary Structure Prediction

The function of an RNA sequence is largely associated with its structure, that is why predicting the RNA structure from its sequence has become increasingly important. The RNA structure formation is mainly a hierarchical process, which can be divided into two steps [19]. The first step is the formation of the secondary structure from a chain of nucleotides. The second step, the formation of the tertiary structure, consists of the bending and folding of the secondary structure, leading to the final three-dimensional fold of the RNA sequence. The tertiary structure of an RNA is more complex and more difficult to obtain experimentally or predict computationally. Because the secondary structure base pairing and base stacking energies already contribute the major part of the energy gained by folding the RNA, the tertiary structure interactions only play a minor role energy-wise and are therefore a lot more difficult to predict. However, the RNA secondary structure is often sufficient to perform a successful functional analysis. One of the most common used algorithms for predicting RNA secondary structure is the Zuker algorithm, based on minimization of the free energy (MFE) [20]. Simply said, the Zuker algorithm assumes that the correct structure of the RNA sequence is the one that has the minimum free energy.

Even tough today the most popular structure prediction algorithms, for folding a single sequence, are based on free energy, this method has limitations in practice [21]. For this reason, to achieve the best accuracy, mostly so-called *comparative methods* are used.

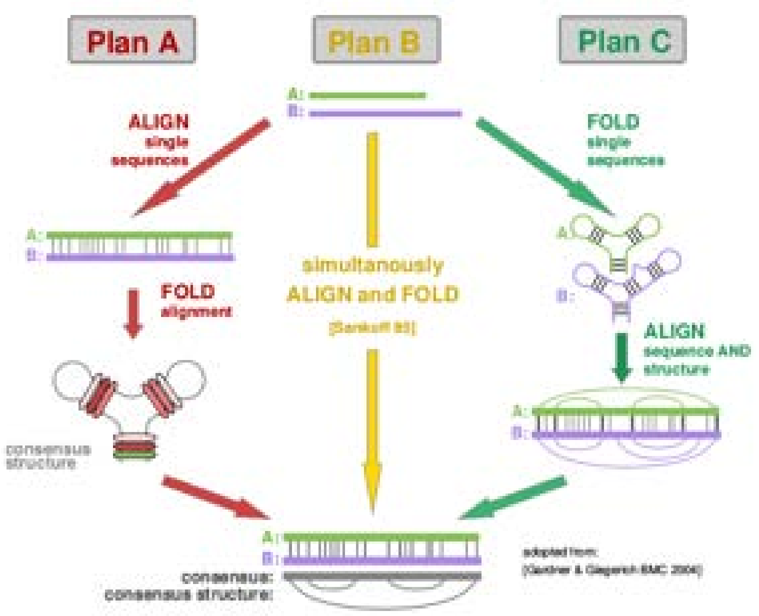


Figure 2.5: Comparative RNA analysis. Plan A at first finds an alignment of the sequences, then a consensus structure. Plan B does simultaneous alignment and structure prediction. Plan C, at first, does structure prediction and then finds the alignment of the structure [18]

2.2.3 Comparative Methods for RNA Analysis

The idea of RNA comparison is - given an input set of sequences - to find the most common structure and sequence, the so-called *consensus structure* and the alignment of these sequences. There are three approaches how to achieve this task, known as Plan A, B and C (Fig. 2.5). In the following sections we will give a brief overview for each plan.

Plan A

Plan A aims to at first align the sequences and then using this alignment, find a consensus secondary structure. The simplest way to align multiple sequences is to use *Progressive Alignment*. Progressive alignment at first makes pairwise alignments (Sec. 2.2.1) for all the sequences, then constructs a guide tree, using these alignments, and finally builds a multiple alignment by following the guide tree. The second step of Plan A is to find a *consensus structure* of the alignment. One of the most famous approaches for predicting consensus structure from an alignment is *RNAalifold* [22].

One of the drawbacks of Plan A is that because the alignment is done before the structure prediction, it can cause structural misalignments. However, it is very useful when sequences have high similarity, or when the alignment is already known.

As we used RNAalifold in this work, we will explain it in more detail later in Sec. 2.3.1.

Plan B

The idea of Plan B is to simultaneously align RNA sequences and predict RNA secondary structures. The bottleneck of this approach is that for a good secondary structure a good sequence alignment is needed, and for a good alignment good secondary structure is needed. However, the *Sankoff algorithm* [23] solves this problem. Nowadays there are many implementations for simultaneous alignment and folding, based on variants of the Sankoff algorithm with heuristics. One example is *LocARNA* [24] which we used in this work and which will be described it in more detail in Sec. 2.3.2.

Plan C

In contrast to Plan A, Plan C at first predicts secondary structure for each single sequence and then computes an alignment for the structures (aligns structures). This approach is useful when there was no information about sequence conservation observed. Many algorithms for predicting RNA secondary structures from single sequence exists e.g. Zuker, Nusinov etc [20]. In Plan C the most interesting part is how the alignment of the predicted structures is done. RNA structures are often represented as trees, due to their nested nature. For aligning sequences edit operations are used for similarity measurement. However it is also possible to generalize this method for trees [25][26]. The idea behind is to construct an RNA tree (Def. 2.2.2) from the secondary structure and then align these trees.

Definition 2.2.2. *An RNA tree is an ordered, labeled tree G, where each node $v \in V_G$ is either a base node, base pair or root. Labels are $l(v) \in \{A, C, G, U\}$ for base nodes and $l(v) \in \{AU, UA, CG, GC, GU, UG\}$ for base pair nodes.*

Just like sequence alignment, tree alignment is the transformation of an RNA tree into the another one, by making use of edit operations to find a common subtree. For tree alignment, basic edit operations are renaming base nodes, insertion/deletion of base nodes, renaming base-pair nodes, insertion/deletion of base-pair nodes. A definition for RNA tree alignment is given in Def. 2.2.3.

Definition 2.2.3. *The alignment of an RNA tree is an ordered, labeled tree T, where each node $v \in V_T$ is either a base node, base pair or root. Each node has a pair of labels $(l_1(v), l_2(v))$ such that:*

- $l_i(v) \in \{\ A,\ C,\ G,\ U,\ -\}$ *for base nodes*
- $l_i(v) \in \{\ AU,\ UA,\ CG,\ GC,\ GU,\ UG,\ --\}$ *for base pair nodes,*

where $i = (1, 2)$.

The crucial point in plan C is the question, whether the initial folding produces at least some structures that can align well, thus give some ideas about the consensus structure if such exists.

2.3 Methods and Technical Overview

In the following sections we will talk about computational tools, frameworks and techniques that were used in the scope of this work.

2.3.1 RNAalifold

RNAalifold finds an optimal consensus structure by minimizing the combination of the *conservation score* and sum of free energy for all RNA sequences present in the alignment. Conservation score γ is computed for each base pair by awarding mutations and penalizing non-complementarities. Eq. (2.3) shows how the conservation score is computed. Function $h(x, y)$ is called *Hamming distance.* It is equal to 0 if $x = y$, or 1 otherwise.

$$\gamma(i,j) = \sum_{1 \le l < l' \le K} \begin{cases} h(\alpha_{li}, \alpha_{l'i}) + h(\alpha_{lj}, \alpha_{l'j}) & \text{if } \alpha_{li} - \alpha_{lj}, \alpha_{l'i} - \alpha_{l'j} \text{ are compl.} \\ 0 & \text{otherwise} \end{cases}$$
$$+\delta \sum_{1 \le l \le K} \begin{cases} 0, & \text{if } \alpha_{li} - \alpha_{lj} \text{ are compl} \\ 0.25, & \text{if } \alpha_{li} - \alpha_{lj} \text{ are both gaps} \\ 1, & \text{otherwise} \end{cases} \tag{2.3}$$

RNAalifold uses four recursive matrices shown in Eq. (2.4), Eq. (2.5), Eq. (2.6) and Eq. (2.7).

$$F_{i,j} = min(F_{i+1,j}, min_{i<k\leq}j(C_{i,k}, C_{k+1,j})) \tag{2.4}$$

$$C_{i,j} = \beta\gamma(i,j) + min \begin{cases} \sum_{\alpha \in A} eH(i,j,\alpha) \\ min_{i<k<l<j}(\sum_{\alpha \in A}(eSBI(ij,kl,\alpha) + C_{k,l})) \\ min_{i<k<j}(M_{i,k} + M^1_{k+1,j} + a) \end{cases} \tag{2.5}$$

$$M_{i,j} = min \begin{cases} M_{i+1,} + c \\ min_{i<k<j} C_{i,k} + M_{k+1,j} + b \\ M^1_{i,j} \end{cases} \tag{2.6}$$

$$M^1_{i,j} = min(M^1_{i,j-1} + c, C_{i,k}) \tag{2.7}$$

Each matrix holds energies of optimal fold for every subsequence $(i,\ j)$ in the following way:

- F - unconstrained structures
- C - structures enclosed by (i,j)
- M - multi-loop components
- M^1 - multi-loop with one branch

$eH(i,j,\alpha)$ and $eSBI(ij,kl,\alpha)$ correspond to *Turner* energy for *hairpin* and *stacking, bulge, internal-loop* enclosed by alignment positions (i,j) and (i,j,k,l) in sequence α of alignment A respectively. In Eq. (2.5), Eq. (2.6) and Eq. (2.7) a, b and c represent contributions for *closing* multi-loop, branch and unpaired positions respectively [22].

2.3.2 LocARNA

LocARNA is a tool to compute alignment of RNA sequences. As described in Sec. 2.2.1, an alignment is basically a comparison of RNA sequences. LocARNA does this comparison by taking into account RNA sequences, as well as secondary structures. The difficulty of such a structural comparison is that each RNA sequence has potentially many structures that it can fold to. LocARNA is using a probability based model, that allows it to consider the most potential structures according to their probability and identify the one that fits the best to all compared RNA sequences *simultaneously.* The idea behind the probabilistic model is to take into account base pair probabilities during the alignment step. By this it is possible to sort out insignificant base pairs from the beginning. As an input, LocARNA requires only RNA sequences and it outputs a multiple alignment together with a consensus structure [24]. Moreover, LocARNA package contains several tools and algorithms which reduce LocARNA runtime complexity. *ExpaRNA-P* [27] and *SPARSE* [28] are examples of such algorithms.

2.3.3 Infernal

Infernal is a suit of tools that can be used to search sequence databases for homologs of structural RNA sequences and to do sequence- and structure-based RNA sequence alignments [29]. Infernal uses a position specific scoring system for edit operations to build a profile from a multiple sequence alignment. Profiles in Infernal are called *covariance models* which are probabilistic models and a special type of a stochastic context-free grammar (Def. 2.3.1) [30]. In other words, covariance models or *CMs* are statistical models of structurally annotated RNA multiple sequence alignments and structures. CMs are closely related to *Hidden Markov Models* (HMMs), but unlike them in CMs base paired positions are dependent on each other. This dependency allows the profile to model covariance at the dependent positions, which often occurs between base paired columns of structural RNA alignments [29].

Definition 2.3.1. *A stochastic context free grammar (SCFG) consists of the following:*

- *Set M of non-terminal symbols or just states.*
- *Set K terminal symbols (e.g. the alphabet (a, c, g, u) for RNA).*
- *A number of production rules of the form: $V \to \gamma$, where γ can be any string of non-terminal and/or terminal symbols, including a special case, the empty string ε.*
- *Each production rule is associated with a probability, such that the sum of the production probabilities for any given non-terminal V is equal to 1.*

In this work we used *cmbuild* and *cmsearch* programs of Infenral. For this reason, in following sections we will talk only about these two programs i.e. how to build covariance models and how to search for homologues.

cmbuild

With *cmbuild* a covariance model for a multiple sequence alignment is built. The only requirement is the consensus secondary structure annotation in the alignment file, because it is used to determine the architecture of the covariance model. As mentioned before, a covariance model is a special type of SCFG. It consists of groups of model states, which are associated with base pairs and single-stranded positions in an RNA consensus secondary structure. A CM has seven types of states and production rules as shown in Tab. 2.1.

State Type	**Production**	**Emission**	**Transition**
P - pair emmiting	$P \rightarrow aYb$	$e_v(a, b)$	$t_v(Y)$
L - left emitting	$L \rightarrow aY$	$e_v(a)$	$t_v(Y)$
R - right emitting	$R \rightarrow Ya$	$e_v(a)$	$t_v(Y)$
B - bifurcation	$B \rightarrow SS$	1	1
D - delete	$D \rightarrow Y$	1	$t_v(Y)$
S - start	$S \rightarrow Y$	1	$t_v(Y)$
E - end	$E \rightarrow \varepsilon$	1	$t1$

Table 2.1: Covariance model: states and rules

Where $Y \in M$ and $a, b \in K$. A covariance model consists of many states, where each state is one of these seven basic types. Each state has its own emission and transition probability distributions, and its own set of states that it can transition to. P states are used to model consensus base pairs, L and R states are used to model single stranded left and right residues respectively. Deletions relative to the consensus are modeled by D states and the branching topology of the RNA secondary structure by B, S, and E states.

The first step of building a CM is to produce a binary guide tree from the consensus structure. The nodes of the tree represent the consensus secondary structure. So the guide tree is a parsing-tree for the consensus structure, with nodes as non-terminals and alignment columns as terminals. The nodes of the guide tree are numbered in preorder traversal (thus parent nodes always have lower indices than their children). We will not go in details how the guide tree is constructed, however important to note that the guide tree is the skeleton on which the covariance model is built [29]. To construct a covariance model from a guide tree set of transitions were introduced. These transitions were designed such that given the guide tree, there is unambiguously one and only one parsing tree for any given individual structure. The final covariance model is an array of M states, connected as a directed graph by transitions $t_v(y)$. One can imagine the covariance model as an array of states in which all transition dependencies run in one direction [29].

cmsearch

cmsearch is used to search one or more covariance models against a sequence database and determine sequences with the most significant matches to the covariance models. Significance is measured either by *cm-bit score* or *E-value*. Cm-bit score is a log-odds score defined as in Eq. (2.8).

$$S(seq, CM) = log_2 \frac{P(seq|CM)}{P(seq|null)} \tag{2.8}$$

Where $P(seq|CM)$ is the probability of the target sequence, according to the covariance model build by *cmbuild*. $P(seq|null)$ is the probability of the target sequence given the *null hypothesis*. In this context the null hypothesis is a one-state covariance model that states that random sequences are independent and identically distributed with equally probable nucleotide composition. So a positive score S indicates that the covariance model is a better model of the target sequence than the *null model*. The E-value represents the expected number of false positives at, or above the specified bit score. Thus the lower the E-value is, the better are the results. To be able to use E-values, covariance models must be calibrated using Infernal's *cmcalibrate* program. *cmsearch* is a pipepline, consisting of several filtering steps. These steps can be divided in two groups: HMM filtering and covariance model filtering. Because covariance models are more complex than HMMs, covariance model filters are used after HMMs (to work only with the sequences that survived HMM filters). The sequences that passed all the filters are reported at the end as homologue sequences for the given covariance model [29].

2.3.4 NSPDK

Graph kernels are used to compute a similarity measure between graphs in terms of dot product. A *decomposition kernel* is a composite kernel that operates over all possible units defined by a specific relation [31]. In this project a special type of decomposition kernel, so-called *Neighborhood Subgraph Pairwise Distance Kernel*(NSPDK) [32], was used. NSPDK defines pairs of subgraphs as *neighborhood* subgraphs (Def. 2.3.2).

Definition 2.3.2. *For a given graph $G = (V, E)$ and an integer $r \geq 0$ the neighborhood subgraph is a subgraph of G with root vertex v and induced by the set of vertices at distance d, where the distance $d \leq r$. Such subgraph is denoted as $N_r^v(G)$.*

When the distance between the roots of two neighborhood subgraphs of radius r is equal to d the *neighborhood-pair* relation $R_{r,d}$ is held. Decomposition kernel on that relation $R_{r,d}$ is defined as Eq. (2.9)

$$k_{r,d}(G,G^{'}) = \sum_{\substack{A,B \in R^{-1}_{r,d}(G) \\ A',B' \in R^{-1}_{r,d}(G')}} \mathbf{1}(A \cong A^{'})\ \mathbf{1}(B \cong B^{'}) \tag{2.9}$$

$$K(G,G^{'}) = \sum_{r}\sum_{d} k_{r,d}(G,G^{'}) \tag{2.10}$$

where the inverse relation $R^{-1}_{r,d}$ indicates all possible pairs of neighborhood subgraphs of radius r with root verticies at distance d in the given graph G. $\mathbf{1}$ represents the indicator function and $\cong$ stands for isomorphism between the graphs. NSPDK is defined as the sum of all kernels for all radii and all distances (Eq. (2.10)). To overcome the high computational costs of similarity and isomorphism tests, authors designed an efficient graph serialization procedure to reduce two isomorphic graphs to an identical string. This was achieved by using a distance information between pairs of vertices. At the end an iterative hashing procedure is used to map the string encoding into an integer code. Thus, the isomorphism test between to graphs is reduced to the equality check between their integer codes [32].

2.3.5 MinHash Technique

Heyne and Costa [1] adapted Minimum Hash function, local sensitive hash function, introduced by Broder [33] to the subject of nearest neighbors and dimension reduction. The purpose of the adaption was to efficiently determine candidate clusters for *GraphClust* pipeline. The main idea was to define the clustering problem in terms of approximate nearest neighbors queries, which in their turn can be answered efficiently. Definition of such a query is given in Def. 2.3.3.

Definition 2.3.3. *Let $P = \{p_1, ...p_n\}$ be a set of n instances in a metric space X with a distance function d. A procedure that returns the instance in P closest to the query instance $q \in X$ is called neighborhood query.*

$$h_i = argmin_{x_j \in x} f_i(x_j) \tag{2.11}$$

The adapted MinHash function is defined as shown in Eq. (2.11) where $f_i : \mathbb{N} \rightarrow \mathbb{N}$ is a set of random hash function. These hash functions must be independent and satisfy following constraints:

- $\forall x_j \neq x_k, f_i(x_j) \neq f_i(x_k)$

- $\forall x_j \neq x_k, P(f_i(x_j) \leq f_i(x_k)) = 0.5$

MinHash function returns the first feature indicator under a random permutation of the features order. The authors notice that the MinHash collision is an unbiased estimator of Jaccard similarity, meaning that the probability of the hush function with minimum hash value for two instances being the same function, is exactly the fraction of non-zero features that these two instances are having in common (Eq. (2.12)).

$$P(h_i(x) = h_i(z)) = \frac{|x \cap z|}{|x \cup z|} = s(s, z) \tag{2.12}$$

To obtain an efficient neighbor search procedure, at first all results from the set of MinHash functions are collected in so-called *instance sketch* in form of tuple $((h_1(x), ..., h_N(x)))$. Then inverse index is built, returning all the instances that have the same MinHash value. Formally saying, for a given i-th hash function and a value $\overline{h} = h_i(x)$, the set of returned instances will be $Z_i(\overline{h}) = \{z \in P$, where $h_i(z) = \overline{h}\}$. Finally the approximated neighborhood Z is induced from the multi-set $Z = \{Z_i\}_1^N$. At the end the elements in Z are sorted according to their occurrence frequency. So *k-neighborhood* of the instance x is the set of k closest elements: $N_k(x)$. Candidate clusters are finally obtained from the densest neighborhoods. The density of the neighborhood for the instance x is defined by the average pairwise similarity between x and all the elements in its k-neighborhood [1].

3 Galaxy Framework

Nowadays it is hard to imagine everyday life without computers, even harder, if not impossible, to imagine science without computational power. Life sciences are not exceptions. Fast developing computational resources give more and more opportunities for new research. However, not every life scientist has programming or computer science skills, which yields to problems during an installation and usage of needed tools. Hence, the accessibility problem of computational tools must be addressed, or in other words, a solution must be provided which will enable scientists with no programming experience to easily integrate and run even complicated tools. Another major aspect for scientific research is reproducibility. Reproducing experimental results requires knowledge about, for example, the current set of tools, parameters and datasets that were used during the experiment. In this case again, the lack of programming skills makes it hard to find out all the details needed for reproducing the experiments.

Galaxy, an open web-based platform, was designed to address these problems. It enables users to perform computational analysis of genomic data [2]. Galaxy is *"bridging the gap between tool developers and researchers"* [34]. It contains various tools needed for analyzing and visualizing data, as well as it gives users access to publications, experiments etc.

In the following sections we will describe the Galaxy framework more detailed. This includes its interface, usage and the integration of a new tool. Furthermore we will explain how Galaxy ensures accessibility and reproducibility of its tools and experiments.

3.1 User Interface

The Galaxy public server gives access to numerous tools for analysis, genomic data and other resources to anyone who has Internet [35]. The Galaxy interface or, so-called, workspace is divided into four parts - top, left, central and right (Fig. 3.1). The navigation bar is located on the top. The left part is so-called *tool panel*, where different tools are located. The tools are grouped in categories

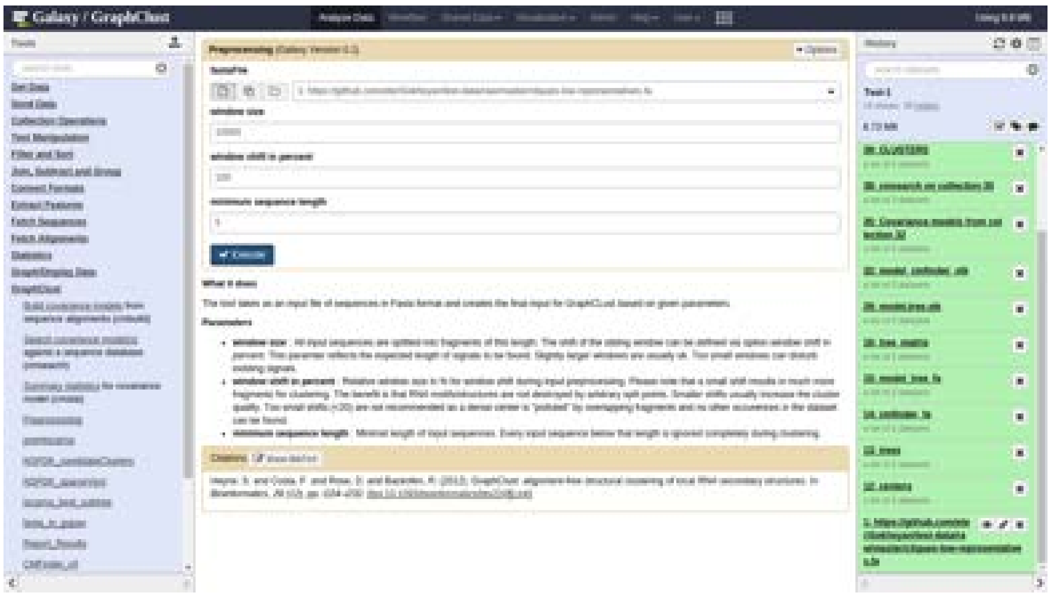


Figure 3.1: Galaxy Workspace. Galaxy workspace is divided in four parts. *Top:* navigation bar. *Left:* tool panel. *Central:* tool interface. *Right:* history.

(e.g. text manipulation, statistics, filters and sorting) and are searchable. The central panel is designed for the tool interface, meaning when users choose a tool from the tool panel, it appears in the central panel. Here inputs and parameters are set for the execution. Finally, the right panel is the history panel, where, all the actions the user performed are stored. This includes the used datasets, tools, information about used parameters and when the tool was executed. Galaxy also provides *interactive tours*, which explain step-by-step how the framework can be used. Galaxy developers or contributors can write such tours for any action they want to explain.

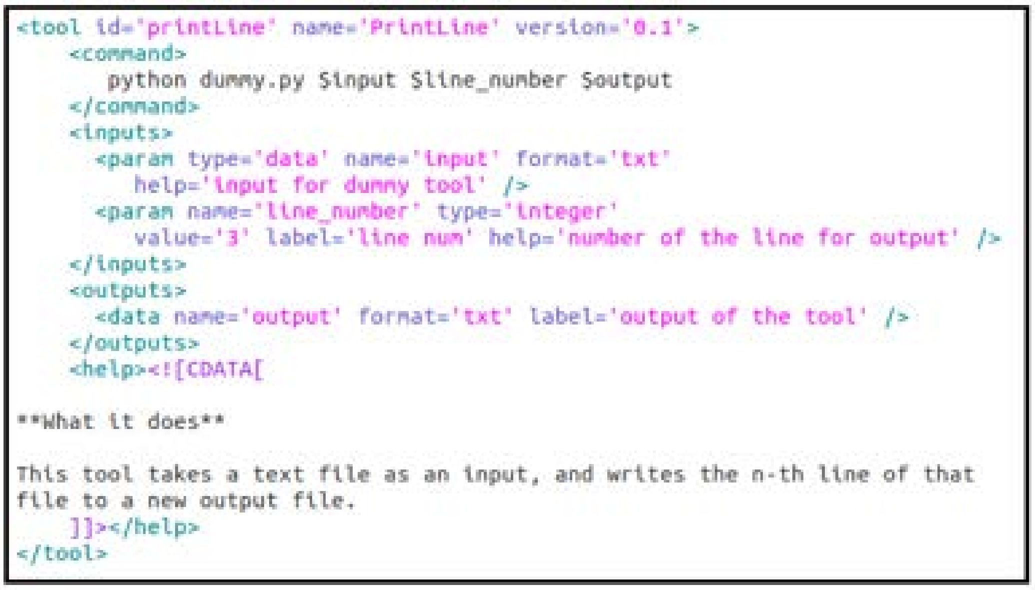
3.2 Tool Wrapper

A Galaxy tool can be any piece of software written in any programming language. To include a tool in the Galaxy platform, one needs to create a tool *wrapper*. The wrapper is an XML file which contains instructions for Galaxy about how to use the tool. Each tool needs its own wrapper. The wrapper must explain what kind of input the tool expects, which output it produces and what parameters it uses. In the dummy example we show how to write such a wrapper. Lets assume we have a tool (e.g. a python script) called *PrintLine* that takes as an input a text file and writes one line, specified by the user, to an output file.

The wrapper for our tool is called *printLine.xml*. Every wrapper in Galaxy must contain a *<tool></tool>* tag. Everything else must be written inside this tag. Mandatory parameters for the *<tool>* tag are *id, name and version*. Once these are defined we can start to describe our tool. As mentioned, the input for our tool is a text file, so within *<inputs></inputs>* tag we tell Galaxy that we need a file in *txt* format by adding the following line:

```
<param type="data" name="input" format="txt" help="input for the tool"/>
```

In the same way we define the parameter for choosing the line to print. In this case the type of the parameter will be *integer*, instead of *data*, and we should also specify a default value for it. The output of the tool is defined similarly to an input, but is written within an *<outputs></outputs>* tag. When all the inputs, parameters and outputs are defined we have to tell Galaxy how to run the tool. The tag *<command></command>* is designed exactly for this purpose. There we just call our tool and give it the needed parameter names starting with a dollar sign ($). The complete wrapper is shown in Fig. 3.2.



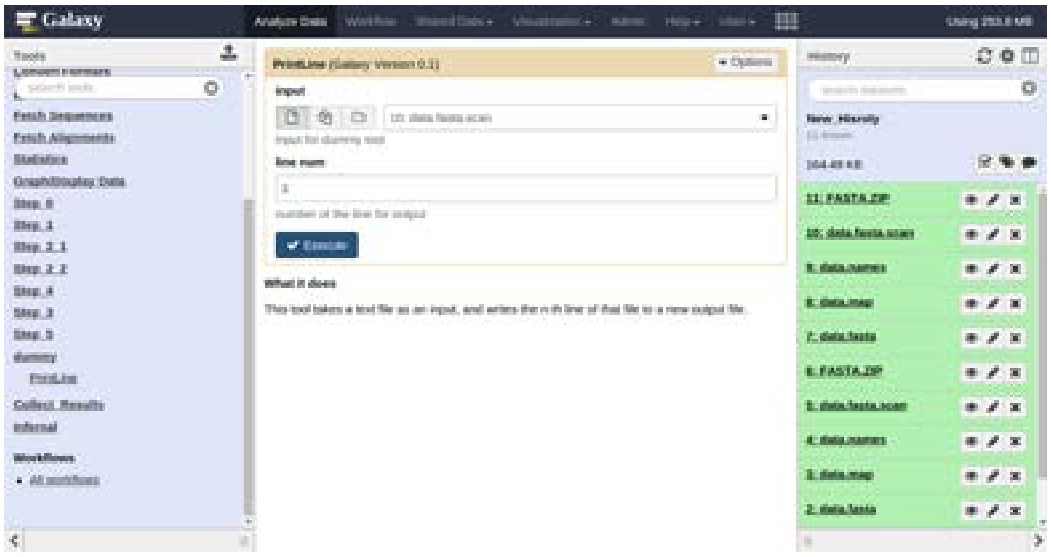
```
<tool id='printLine' name='PrintLine' version='0.1'>
    <command>
      python dummy.py $input $line_number $output
    </command>
    <inputs>
      <param type='data' name='input' format='txt'
         help='input for dummy tool' />
      <param name='line_number' type='integer'
         value='3' label='line num' help='number of the line for output' />
    </inputs>
    <outputs>
      <data name='output' format='txt' label='output of the tool' />
    </outputs>
    <help><![CDATA[

**What it does**

This tool takes a text file as an input, and writes the n-th line of that
file to a new output file.
    ]]></help>
</tool>
```

Figure 3.2: printLine.xml Galaxy wrapper for a dummy tool. The tool takes as an input a text file and writes one line, specified by the user, to an output file.

The *<help></help>* tag - even tough not mandatory - is a very useful tag. With it developers can write detailed descriptions of their tool and make it more user friendly. These are the very basic and necessary steps for the wrapper creation. Galaxy provides very detailed online tutorials for the tool creation and integration of new tools into Galaxy [36]. After the tool is integrated into Galaxy it will appear in the tool panel. Fig. 3.3 shows how the tool looks like in the Galaxy workspace.



PrintLine (Galaxy Version 0.1)

Input: 10: data.fasta.scan

Input for dummy tool

line num: 3

number of the line for output

What it does

This tool takes a text file as an input, and writes the n-th line of that file to a new output file.

Figure 3.3: Integrated PrintLine Tool. All tools integrated in Galaxy have a uniform graphical interface.

In the tool panel, on the left, we created a new category called *dummy* and put our tool there. By clicking on the PrintLine tool, in the central panel we can see the interface of our tool. As we described in our wrapper, here we must specify the input file and the number of the line we want to get. By clicking the *execute* button, our tool will run and give us the desired output, which will appear on the history panel.

3.3 Accessibility in Galaxy

While introducing our tool to the Galaxy framework via the wrapper, in the previous section, we did not write any style information, for the tool appearance. However, Galaxy was able to graphically show the tool. The reason is that, Galaxy automatically generates a web interface which is uniform for the all integrated tools. This means that all tools interfaces have the same style, which makes it easier for users to work with new, unfamiliar tools. This and the easy integration of the new tools improves accessibility. Another aspect that makes the Galaxy framework accessible is the data. Users can easily upload their own data to the server, or use already available data from several established data warehouses. Fig. 3.4 shows a part of the data acquisition options on the tool panel of Galaxy [37].

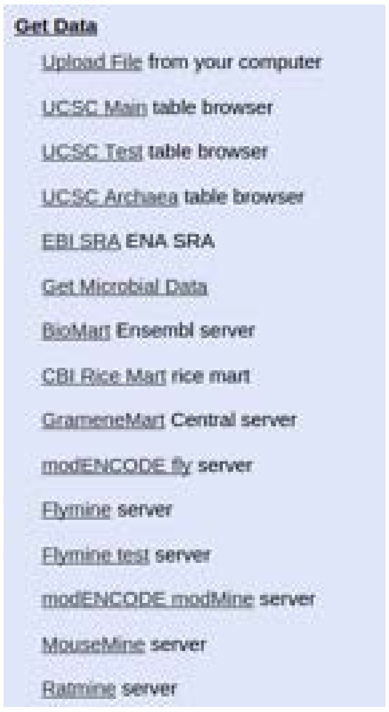
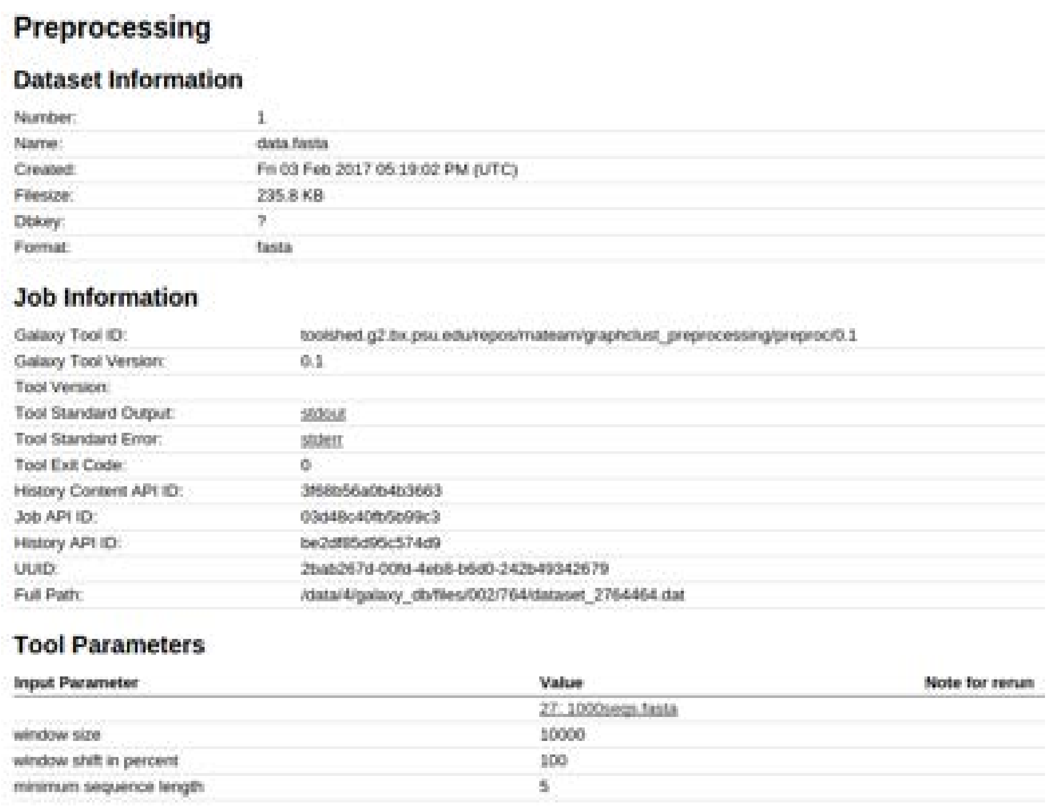


Figure 3.4: Galaxy Datasets. A part of the data acquisition section. Galaxy users can import their own datasets, but also make use of the data provided by warehouses.

3.4 Reproducibility in Galaxy

Galaxy enables users to import datasets and use different tools for analyzing these datasets. Reproducing these analyses can be helpful for understanding the results and thus, for better usage of them. It was mentioned earlier, that in order to be able to reproduce an experiment or analysis, a detailed description capturing all aspects of an analysis, including used tools, datasets and parameters is needed. Galaxy automatically tracks this kind of information every time a tool or set of tools is used. This is called *metadata* in Galaxy. The metadata is stored in a relational database, which is available in the history panel after the execution of the tool is finished. These histories can be viewed, copied and even shared with other users. Examples of such metadata are shown in Fig. 3.5. Moreover, Galaxy's interface provides an option to automatically re-run the performed task with the same parameters, without manually setting them again. Another aspect that supports reproducibility is *user created metadata*. Galaxy supports labeling of items and annotations. By labeling and annotating the items, users can provide better understanding of the purpose of the performed actions to other users [37].



Preprocessing

Dataset Information

Number:	1
Name:	data.fasta
Created:	Fri 03 Feb 2017 05:19:02 PM (UTC)
Filesize:	235.8 KB
Dbkey:	?
Format:	fasta

Job Information

Galaxy Tool ID:	toolshed.g2.bx.psu.edu/repos/mateam/graphclust_preprocessing/preproc/0.1
Galaxy Tool Version:	0.1
Tool Version:	
Tool Standard Output:	stdout
Tool Standard Error:	stderr
Tool Exit Code:	0
History Content API ID:	3f68b56a0b4b3663
Job API ID:	03d48c40fb5b99c3
History API ID:	be2df85d95c574d9
UUID:	2bab267d-00fd-4eb8-b6d0-242b49342679
Full Path:	/data/4/galaxy_db/files/002/764/dataset_2764464.dat

Tool Parameters

Input Parameter	Value	Note for rerun
	27: 1000seqs.fasta	
window size	10000	
window shift in percent	100	
minimum sequence length	5	

Figure 3.5: Galaxy Tool Metadata. Galaxy metadata contains detailed description of the tool, including the used parameters, datasets and the execution time.

3.5 Workflows

Usually data analysis consists of several steps, this means scientists make use of several consecutive tools to achieve desired results. This procedure can be time-consuming, when one has to run the same experiment many times. Galaxy *workflows* help users to avoid this inconvenience. One can think about Galaxy workflow as a rooted, directed, cyclic graph. The nodes of the graph are the tools, represented as rectangles and divided in two sections for inputs and outputs. The edges of the graph are connections between these tools. In this context, a connection defines the output of a tool which should be used as an input for another tool. The root of the graph is the input that will be passed to the following tools for execution. Galaxy provides a graphical workflow editor, where users can easily create workflows. Similarly to the Galaxy workspace, the workflow editor is divided in four parts. The navigation bar is on the top, where users can switch between the workflow editor and the workspace. The left-most panel is the tool panel, but unlike in workspace, here the tools can not be executed. Instead, users can select, drag and drop

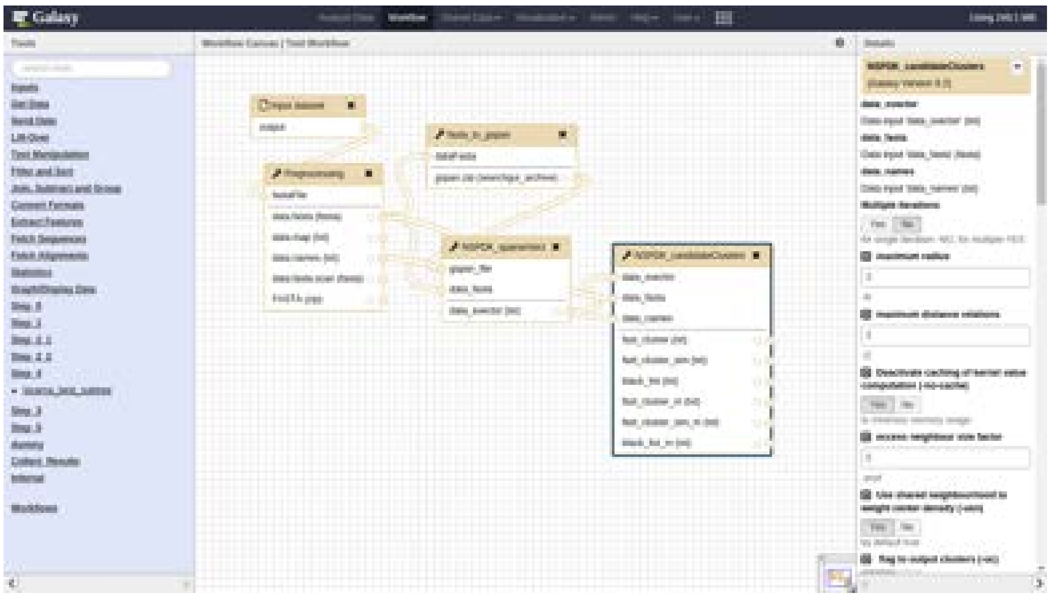


Figure 3.6: Galaxy Workflow Editor. The workflow editor is divided in four parts. *Top*: navigation bar. *Left:* tool panel. *Central:* editor panel. *Right:* parameter panel.

the needed tool into the central panel, which is the editor panel. To connect two tools together, users should select and drag the output from one tool to the input section of another one. However, to be able to connect two tools, the output format of one tool must the same as the input format of the other one. The right-most panel is the tool parameter panel, where users can define the parameters for each tool separately. Once the workflow is created and saved, it will appear in the workflow section of the Galaxy workspace. Now it can be used as any other single tool. Every time a workflow runs, it runs the same tools, in the same order with the same parameter configuration. However, like for any other tool, the default parameters of the workflow can be changed. When the workflow is executed, not only the final results are stored in the history, but all intermediate results as well. However, it is possible to hide these intermediate results from the history and automatically clean them up. And as with single tools, metadata is generated for a workflow, containing detailed information about every single step. Fig. 3.6 shows an example of a workflow composed of four tools in the workflow editor.

In addition, Galaxy enables users to chain workflows. Basically, to have a workflow containing one or more workflows. Such workflows are called *sub workflows*. In other words, a workflow is called sub workflow if it is a part of another workflow. As mentioned above, after creation, workflows appear in Galaxy tool panel, therefore in workflow editor, instead of a single tool users can choose existing workflow to add to their own workflow.

4 GraphClust : Challenge and Approach

In recent years, non-coding RNAs became a key research topic in molecular biology. New discoveries are quickly changing our impression of genome complexity and raising more challenging questions. That is why the comparison, prediction and functional annotation of non-coding RNAs are now one of the most important tasks of modern RNA research. However, precise annotations for the majority of predicted non-coding RNAs still remain difficult to achieve. One of the reasons for that is that non-coding RNAs, unlike protein coding ones, are divided in various classes, with a huge number of different structures, functions and evolutionary patterns [17]. An RNA class contains such non-coding RNAs that share common structural and functional properties, even though they do not have obvious homology at the sequential level. Thus, clustering RNA sequences according to sequence–structure similarity has now become a generally accepted scheme for ncRNAs annotation.

In 2012, Steffen Heyne and Fabricio Costa presented a novel approach for clustering RNA sequences, according to sequence and structure information: *GraphClust* [1]. GraphClust is a tool for alignment-free structural clustering of local RNA secondary structures. It was proven to have linear run-time and thus is able to work with up to hundreds of thousands of sequences.

In the following sections we will describe the existing GraphClust pipeline in details. Furthermore we will highlight problems the pipeline is facing now and introduce our approach, designed to solve these problems.

4.1 GraphClust Pipeline

As mentioned above, GraphClust is a pipeline for clustering RNA sequences using sequence and structural information. The authors propose to first sample a small number of probable, but adequately different structures for each RNA sequence and encode these structures as labeled graphs. In other words, the structure of the sequence is represented as a graph with possibly disconnected components. Then they extract an explicit vector representation for each graph and build an inverse index on a compressed representation using hashing techniques, instead of directly computing the similarities between graphs. This is done to avoid a quadratic number of comparisons and achieve constant runtime for nearest neighbor searches for any target structure. Once neighborhoods are defined, they are evaluated and the neighborhoods that contain very similar elements are chosen as candidate clusters. To achieve better accuracy, resulting candidate clusters are refined using alignment techniques. Then a covariance model is constructed for each cluster. In the final step, the dataset is scanned using the constructed covariance model and hits are added to the corresponding clusters. GraphClust is an iterative approach, so the sequences that were already clustered are removed from the dataset and the process starts again [1].

Fig. 4.1 represents the nine steps of the pipeline. In the following sections we will describe each step in detail.

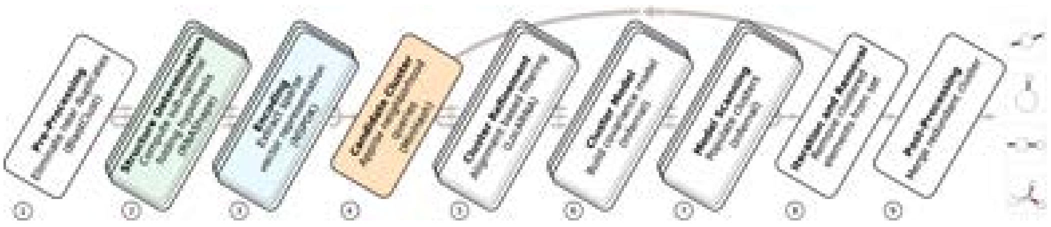


Figure 4.1: GraphClust pipeline. The GraphClust pipeline is an iterative approach for clustering RNA sequences. It consist of nine steps. The iteration performed for steps (4) to (8) [1].

Preprocessing

The GraphClust pipeline starts with a *preprocessing* step. As an input, GraphClust takes a file in FASTA format. As mentioned before, GraphClust can work with hundreds of thousand of sequences, so the FASTA file can be very big. In this step the *BLASTClust* [38] is used to eliminate nearly identical sequences to achieve better accuracy for clustering. Also long sequences can be split into smaller fragments to enable local signal detections. This step is sequential, but it runs only once and creates several files, e.g. file containing the length of each sequence, new generated IDs and sequences etc. These files will be used by other steps of the pipeline during the clustering procedure.

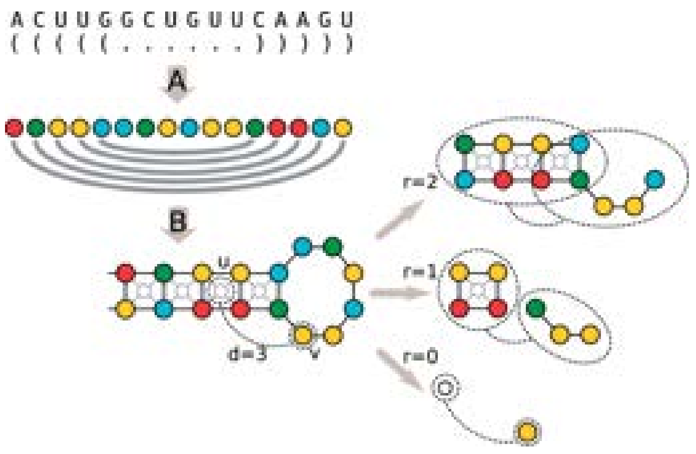


Figure 4.2: RNA secondary structure encoding. The graph encoding preserves the nucleotide information (vertex labels) and the base pairs (edge labels), here shown in different colors (A). Additional vertices are inserted to induce features related to stacking base-pairs quadruplets (B). On the right: example of features induced by the NSPDK for a pair of vertices u, v at distance 3 with radius 0,1,2. Neighborhood graphs are enclosed in dashed ovals [1]

Structure Determination

In this step sequences are divided into smaller overlapping fragments. Then *RNAshape* [39] is used on each resulting subsequence to obtain the l most representative structures for that subsequence. Then each structure is encoded as a graph. Vertices of the graph encode the nucleotides of the subsequence and edges encode nucleotide sequence adjacency information and the pairwise binding status. Also, for each stacking base pairs quadruplet an additional vertex is added to the graph, with four edges connected to the corresponding nucleotides. Fig. 4.2 represents such an encoding of an RNA structure to a graph.

Encoding

Then, the pipeline produces explicit sparse feature encodings, based on the graphs produced in the previous step. To deal with entities represented as graphs, the authors chose the *neighborhood subgraph pairwise distance kernel* (*NSPDK*) (Sec. 2.3.4). In GraphClust the integer code, produced by NSPDK, is used as a feature indicator. In other words, the integer associated to each feature can be interpreted as the feature key and the normalized count of occurrences as its value. By this, a sparse vector in $\mathbb{R}^m$ is obtained for an explicit feature representation for a given graph G.

Candidate Cluster

After a sparse vector is obtained, the candidate clusters can be found using the technique described in Sec. 2.3.5. The most dense instances are more likely to be in the same cluster, thus returning only the top ranking dense neighborhoods would result in highly redundant sets. To overcome this problem, some threshold *th* is chosen, and candidate clusters are chosen as the top ranking neighborhoods, if the size of their overlap is below the *th*. To speed up this phase, a small set of sequences is uniformly randomly extracted. And only sequences in this set are ranked according to their approximate density. This is possible instead of ranking the entire set of sequences, as the larger the cluster, the higher the probability that it will be hit by a sample.

Cluster Refinement

By the end of the previous step, sequences in obtained candidate clusters are considered similar according to the *NSPDK* similarity measure (Sec. 2.3.4). To increase the quality of candidate clusters and to take into account domain specific knowledge (e.g. compensatory mutations) LocARNA [24] (Sec. 2.3.2) is used to compute a sequence-structure alignment for each cluster. LocARNA produces a matrix with pairwise distance information. This matrix is used to create a cluster tree using the average-linkage algorithm. All subtrees that have at least 3 leaves, meaning 3 sequences, are ranked according to their average pairwise distance of the leaves. Only the top ranked subtree is returned.

Cluster Model

For each refined cluster a covariance model is created by using *Infernal* (Sec. 2.3.3). The covariance models are built based on the consensus secondary structure of the multiple sequence alignment defined in the previous step.

Model Scanning

After the covariance model for each cluster is obtained, it is used to search against the whole dataset for potential cluster members. This procedure is done by *cmsearch* tool from *Infernal* package (Sec. 2.3.3). In this step some ambiguity is allowed, meaning that some sequences can be assigned to several clusters.

Iteration and Removal

After sequences are assigned to clusters, they are removed from the dataset and the next iteration starts again from defining new candidate clusters. This iterative process continues until either 1) no more sequences left to cluster, 2) the predefined number of iterations is performed, or 3) some time limit is reached.

Post-processing

After all iterations are finished, similar clusters are merged together and the sequences that were assigned to more than one cluster in this step will be assigned to only one cluster unambiguously. The similarity of the clusters is measured by their pairwise relative overlap. Two clusters are merged together if the overlap is e.g. more than 50%.

4.2 Challenge

In order to use GraphClust, the user at first has to download the source code of the pipeline and compile it. However, for successful compilation, the user has to be sure that all the tools that the pipeline is using, are already installed e.g. Vienna RNA Package, RNAshapes, LocARNA, Infernal and others. Once all needed tools are installed, their paths should be added to the *$PATH environment variable* for GraphClust to find them. Only after that the compilation can be successful, and the user can start exploiting it. GraphClust provides a set of default parameters for running the pipeline, yet the user can adjust these parameters before running it. Because GraphClust has no graphical interface, everything is done through command line interface, it can be tricky to set all the parameters correct. The *config* file, which contains all the parameters that will be used during the whole process of GraphClust, does not contain any information about the purpose and the meaning of the parameters, except the descriptive name. The above mentioned is making the usage of the GraphClust hard for scientists with no or very little programming skills. Which limits the accessibility of the pipeline.

Another limitation of the pipeline is the tight requirements for the third-party tools. As mentioned before GraphClust uses different tools at the different stages of the clustering. The current GraphClust configuration is rather inflexible when it comes to a tool exchange, whether due to maintenance or upgrade. The poor modularity of the pipeline does not allow users to change any third-party tool without going deep into the source code.

Despite these criticisms, the efficiency of the GraphClust pipeline is hard to underestimate. The approach we explain in the following section is designed to overcome these limitations at the same time keep the efficiency of the original pipeline.

4.3 Approach

The key idea of our approach is to make GraphClust more modular, accessible and transparent, and to integrate it into the Galaxy Framework. As described in Ch. 3 Galaxy framework increases accessibility and reproducibility of the integrated tools. Thus, we decided to separate each step of the pipeline by creating independent tools for each step, then create wrappers (Sec. 3.2) for each of these tools, integrate them in Galaxy, and finally build a workflow (Sec. 3.5) to connect all the components. From now on we will refer to our approach as *Galaxy-GraphClust*.

The configuration of the Galaxy framework allows users to avoid manual installation processes for any third-party tool. By defining requirements for third-party tools or libraries in the tool wrapper, developers enable Galaxy to automatically install these tools without any engagement from the user. The only prerequisite for this is the existence of *conda packages* for these tools [40]. During the development of our approach we used already existing conda packages as well as created new ones for the tools that were missing them.

We started the development of our approach by creating a tool for the preprocessing step. The input of the GraphClust pipeline is a file with sequences in FASTA format. The same type of input we expect for our new tool, which we named *preprocessing*. There is no restriction on the length or the number of the sequences. Our tool has the same parameters and functions as the preprocessing step of GraphClust with only one difference. We eliminated the BLASTClust tool from the preprocessing step. As the goal of our project is to integrate GraphClust to Galaxy, and as Galaxy already contains several tools which serve similar purpose as BLASTClust, we decided to not include it in the preprocessing step. This will allow users to choose a tool, for eliminating redundant sequences, that fits the best for their dataset and needs. The new tool has three parameters called *window size*, *window shift* and *minimum length*. All the input sequences will be split into fragments of the length defined by the window size parameter. The window shift parameter represents a percentage by which the windows will overlap. All the input sequences shorter than the defined minimum are ignored. Fig. 4.3 shows an

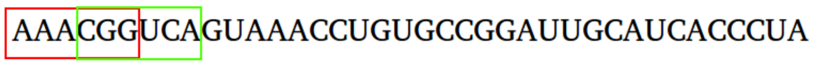
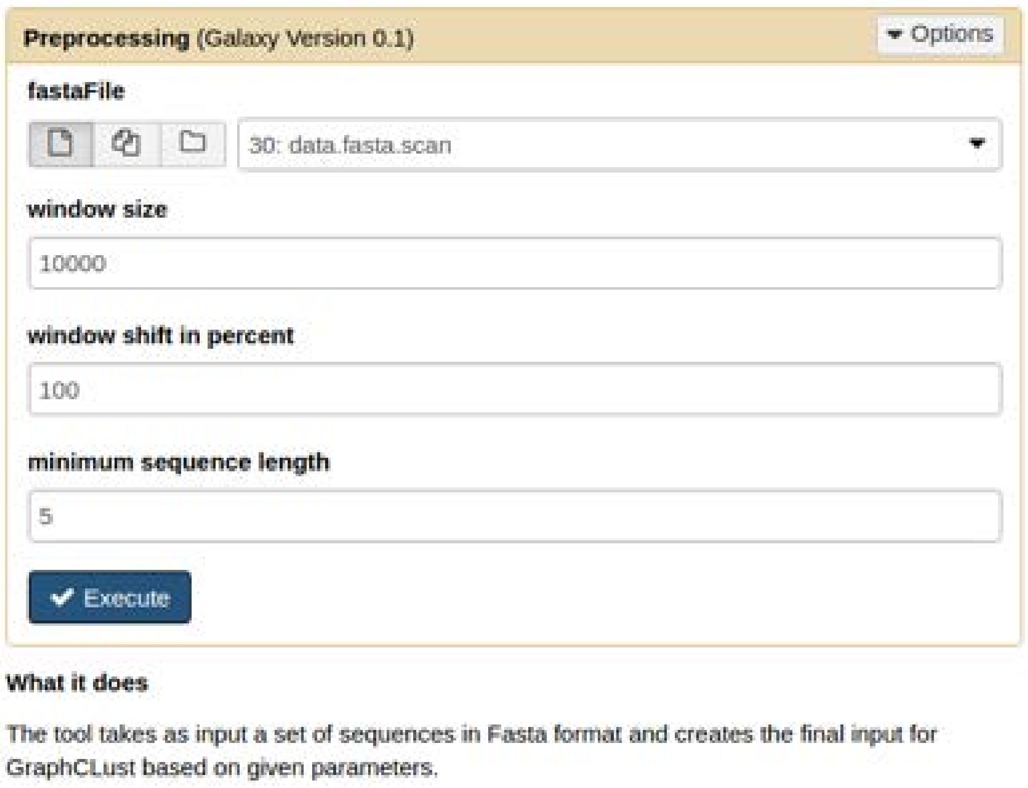


Figure 4.3: Sequence Splitting. The red rectangle highlights the first fragment of the sequence and the green one highlight the second one, where the window size = 6 and the window shift = 50%.

example of the splitting sequences into fragment where the window size is 6 and the window shift is 50%. The red and the green rectangles highlight the first fragment and the second fragments respectively. However, if there is no need to split the sequences one can define very high value for the window size (e.g. 100000) and 100% for the window shift to avoid splitting. The tool produces nine files, which contain information about the input sequences as well as the actual sequences with newly defined unique IDs. These files later are used by other tools. Furthermore, we decided to group all the files and output them as a *zip* file, that users can download and see all the files at one place. The interface of the integrated Preprocessing tool is shown in Fig. 4.4.

The next step of the pipeline is the structure determination. This step requires RNAshapes [41], and because there already was a conda package for it, we simply included it in the requirements of the wrapper. The tool, which we called *fasta_to_gspan*, takes as an input the preprocessed FASTA file. On each sequence of the input, RNAshapes tool is used to construct a set of structures, based on the parameters obtained from Galaxy, and save them in to separate files. Then these structures are converted into graphs and finally grouped together. The final output of the tool is a file containing graph descriptions for all the sequences.
The *encoding* stage is the next step of the pipeline. This and the next step are using NSPDK (Sec. 2.3.4), therefore, we created a conda package for NSPDK to make the deployment easier. The tool, we created, expects two input files. First one is the graph file created in the previous step, and the second one is the preprocessed FASTA file with all the sequences. The tool produces sparse vectors containing explicit feature encodings.

Above mentioned tools are part of so-called *pre-iteration* phase. More specifically, during the whole process of clustering, these three tools will be called only once, they are not part of the iteration. This means also that all the parameters these tools are using are global and they do not depend on the number of iterations. The iterative phase starts with finding candidate clusters. As mentioned above, this is done using NSPDK tool. Thus, the tool we created is called *nspdk_candidateClusters*. This tool requires three input



fastaFile: 30: data.fasta.scan

window size: 10000

window shift in percent: 100

minimum sequence length: 5

What it does

The tool takes as input a set of sequences in Fasta format and creates the final input for GraphCLust based on given parameters.

Figure 4.4: Preprocessing. Graphical interface of the Preprocessing tool integrated into Galaxy. The 'Help' section is not shown completely in the picture.

files. The first input contains the sparse vectors obtained from the previous step, and the other two files are obtained from the preprocessing step. The nspdk_candidateClusters tool contains all the parameters the original pipeline is using for this step. However, here we have an extra parameter, which determines the number of the current iteration. More specifically, the tool has two different configurations for a single iteration and multiple iterations. If the tool is in multiple iteration mode, some additional input files and parameters are needed e.g. *blacklist* and the output of the tool from the previous round.

In a similar manner, we created tools and corresponding wrappers for all the remaining steps. As a result, we got the following tools - *preprocessing, fasta_to_gspan, nspdk_sparseVect, nspdk_candidateClusters, pgma_graphclust, locarna_graphclust, cmfinder, cluster_collection_report*. For building a covariance model and searching homologue sequences we used Infernal 1.1 available in Galaxy. Having the steps of the pipeline as separate tools allows users to not only use the entire pipeline, but also to use separate tools for their own purposes. Furthermore, this level of modularity allows users to easily update or exchange parts of the pipeline if the need occurs.

The cluster_collection_report tool is the very last step of our pipeline. This step is very similar to the post-processing step of the original pipeline, but in our tool we have an extra option. As mentioned before, we excluded BLAST-Clust tool from our approach. However, the configuration of Galaxy enables users to use any tool for preprocessing data before running Galaxy-GraphClust. One of these tools is CD-hit [42]. So, in the cluster_collection_report tool, we included an option that allows us to add sequences to the already defined clusters. More specifically, if CD-hit was used for removing highly similar sequences from the input dataset of the Galaxy-GraphClust pipeline, providing the output of CD-hit to the cluster_collection_report tool, we will have the clusters enriched with the sequences that were removed by CD-hit. The new sequences will be added to these clusters that contain their representaives. Moreover, we added R-scape [43] visualizations for identifying base pairs of the consensus structure with statistically significant covariation. Galaxy's interface allows usage of the in-browser view of the results, which enables a quick inspection of the latter.

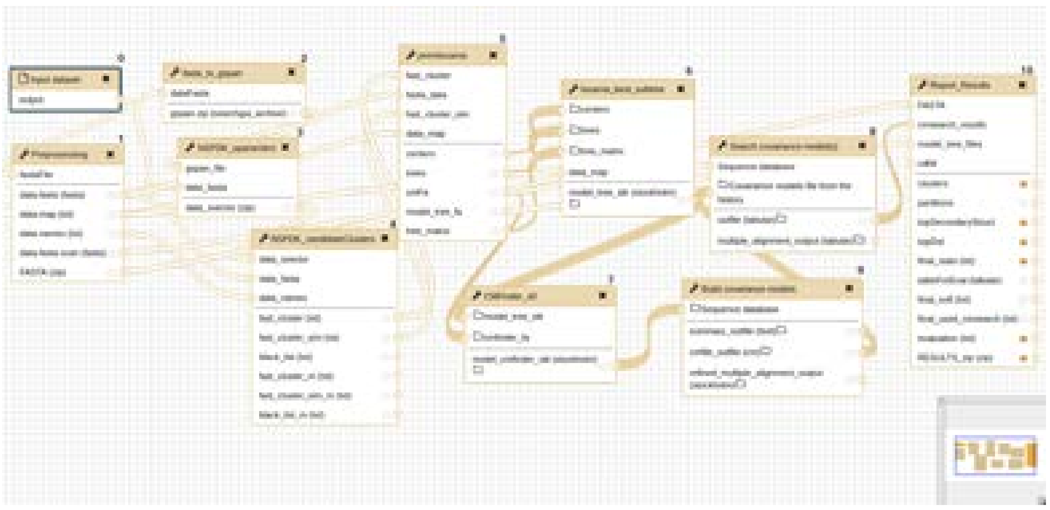


Figure 4.5: Galaxy-GraphClust. 1 iteration workflow of Galaxy-GraphClust in the workflow editor. The numbers at the top of each box are indicating the execution order of the tools.

The final step of our approach is the construction of a workflow (Sec. 3.5). To create simple one round pipeline we just connected all the tools in the correct order (Fig. 4.5). To enable the iterations we decided to create sub workflows. We constructed such a subworkflow for the iterative part of our pipeline and by duplicating and connecting these subworkflows we will have iterations. Galaxy enables users to download, upload and share workflows. The workflow appears in the tool panel in the *Workflows* section. To run a workflow the user should just click on it and execute as any other tool. After clicking on the workflow it is displayed in the tool interface panel, where users can change the parameters of each tool if needed.

As a result, Galaxy-GraphClust is a collection of integrated Galaxy tools, combined in workflows for clustering RNA sequences. Moreover, these workflows can be easily extended by any pre- or post-processing steps. As mentioned before, an example of such an extension can the preprocessing step performed by CD-hit. Additionally, Galaxy-GraphClust supports more recent versions of LocARNA and Infernal packages.

5 Data Analysis

We have tested and measured the performance of Galaxy-GraphClust by clustering known ncRNAs obtained from the Rfam [44] database. We compared our results with results obtained from the original GraphClust pipeline. Furthermore, as a use case for our approach, we worked with a *metatranscriptome* dataset dominated by *cyanobacteria*. In the first part of this chapter we discuss the benchmarking results and then we present a quantitative analysis of the results of clustering the metatranscriptome dataset.

5.1 Benchmarking

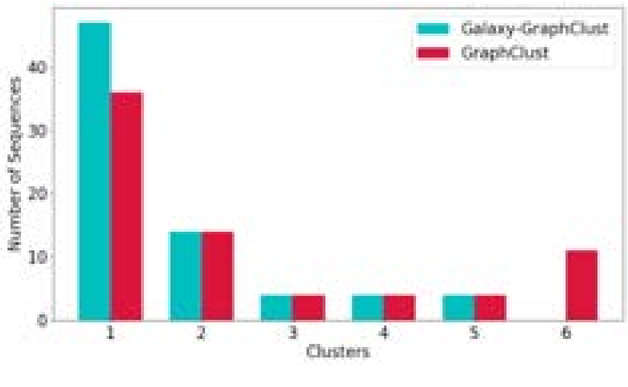
To compare the performance of our approach with the original GraphClust pipeline we performed five clustering rounds on three small datasets of labeled ncRNAs using both approaches. As a sample datasets we used the *Rfam-cliques* [45] benchmark dataset, which is based on Rfam 12 [46] family seed alignments. The Rfam-cliques benchmark dataset consists of 3 different datasets, named *Rfam-cliques Low*, *Rfam-cliques Medium* and *Rfam-cliques High*. Tab. 5.1 shows quantitative information about these three datasets.

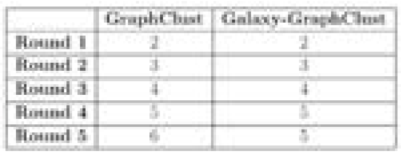
Dataset	Number of sequences	Number of families	Average sequence length
Low	92	10	78 nt
Medium	166	26	87 nt
High	234	48	95 nt

Table 5.1: Rfam-cliques datasets. Quantitative statistics about the size, length and the number of RNA families per dataset.

The parameters for both, GraphClust and Galaxy-GraphClust, were set identically. The only difference is in the version of Infernal. GraphClust uses version 1.0, whereas Galaxy-GraphClust uses version 1.1 of Infernal. The upgraded version of Infenral enables us to use E-values, instead of covariance model scores for the final refinement of the clusters. The preprocessing parameters were chosen in a way to avoid fragmentation of the sequences.

To evaluate the quality of the clusters we used clustering metrics from the *scikit-learn* machine learning library [47]. More specifically, we measured *completeness*, *homogeneity*, *V-measure* and *adjusted Rand score*. The completeness score measures intra-cluster similarity, while homogeneity measures inter-cluster similarity. In other words, completeness states how similar the elements within a single cluster are, whereas homogeneity measures how different clusters are in comparison. The V-measure is the harmonic mean between homogeneity and completeness. These metrics can take values between zero and one, where 1 indicates a perfectly complete and homogeneous clustering. The aim of the adjusted Rand index is to establish an overall comparison between the predicted and the ideal clustering. In other words, the Rand index computes the percentage of pairs of objects for which both clustering methods, the computed and the ideal one, agree. It takes values from -1 to 1, where completely random clustering gets a score close to 0 and perfect match score 1. The adjusted Rand index can yield negative values if the index is less than the expected index. As ground truth class assignments for our sequences, we took the RNA family each sequences belongs to. Having such ground truth enables us to compute the mentioned metrics. However, for the case that not all the sequences of the dataset were clustered, we manually assigned each unclustered sequence to a new cluster, in order to be able to do our evaluation, e.g. if 5 sequences where not clusters we manually add 5 new clusters, each containing one of the unclustered sequences.





	GraphClust	Galaxy-GraphClust
Round 1	2	2
Round 2	3	3
Round 3	4	4
Round 4	5	5
Round 5	6	5

(b)

Figure 5.1: Rfam-cliques Low. **(a)** Predicted clusters after 5 rounds. The height of the bars represents the number of sequences in each cluster. **(b)** Number of predicted clusters after each round.

We started with clustering the Rfam-cliques Low dataset. After five iterations both approaches clustered 73 sequences out of 92. Fig. 5.1 (a) shows the final clusters after the 5th iteration. The blue and red bars represent clusters obtained from Galaxy-GraphClust and GraphClust respectively. Fig. 5.1 (b) represents the number of predicted clusters after each round. As it can be observed in the figure, GraphClust predicted six clusters whereas Galaxy-GraphClust predicted five. In the first iteration both approaches predicted two clusters, however, Galaxy-GraphClust clustered 41 sequences for the first cluster, while GraphClust clustered only 36. The clusters predicted in the next three iterations were identical for both approaches. In the last iteration both approaches predicted one new cluster, which contains sequences from the same family as the cluster found in the first iteration. However, because in the first iteration GraphClust found less elements for that cluster, the overlap percentage was not enough to merge these clusters, which resulted in two clusters containing elements from the same family for GraphClust, while Galaxy-GraphClust merged these clusters. The content of the clusters is shown in Fig. 5.2, where each color represents a specific RNA family. The Fig. 5.2 (a) shows the clusters predicted by GraphClust and (b) the clusters from Galaxy-GraphClust. As it can be seen, cluster 1 and cluster 6 from GraphClust contain elements from the same family, whereas each cluster predicted by Galaxy-GraphClust contain elements from a single family. The evaluation of the clusters shown in Tab. 5.2 illustrates that the clusters obtained from Galaxy-GraphClust have better scores for all four evaluation metrics than the ones from GraphClust.

Similarly, we performed five clustering iterations on Rfam-cliques Medium and Rfam-cliques High datasets. For Rfam-cliques Medium, GraphClust clustered 120 sequences out of 160, resulting in 12 different clusters. In contrast, Galaxy-GraphClust clustered 107 sequences in 9 clusters (Fig. 5.3). For Rfam-cliques High, the number of clustered sequences is 163 in 17 clusters with GraphClust and 140 sequences in 14 clusters with Galaxy-GraphClust (Fig. 5.4). The difference in the number of clusters and the number of sequences between 2 approaches is influenced by difference in versions of Infernal. Infernal 1.1 has several major improvements compared with version 1.0. Besides the huge improvement in the run time, in the newer version the authors changed the definition of insert and match columns for alignments. This and other changes described in [29] mean that for a given input alignment a model built with version 1.1 may have different numbers of states and nodes, and will have usually slightly different parameters, than a model built from the same alignment with version 1.0.

The distribution of RNA families on different clusters for Rfam-cliques Medium and Rfam-cliques High are shown in Fig. 5.5 and Fig. 5.6 respectively. After each iteration both approaches merged overlapping clusters if such existed. However, the quality of predicted clusters and the merging was better for Galaxy-GraphClust, which is indicated by the evaluation scores in Tab. 5.3 and Tab. 5.4.

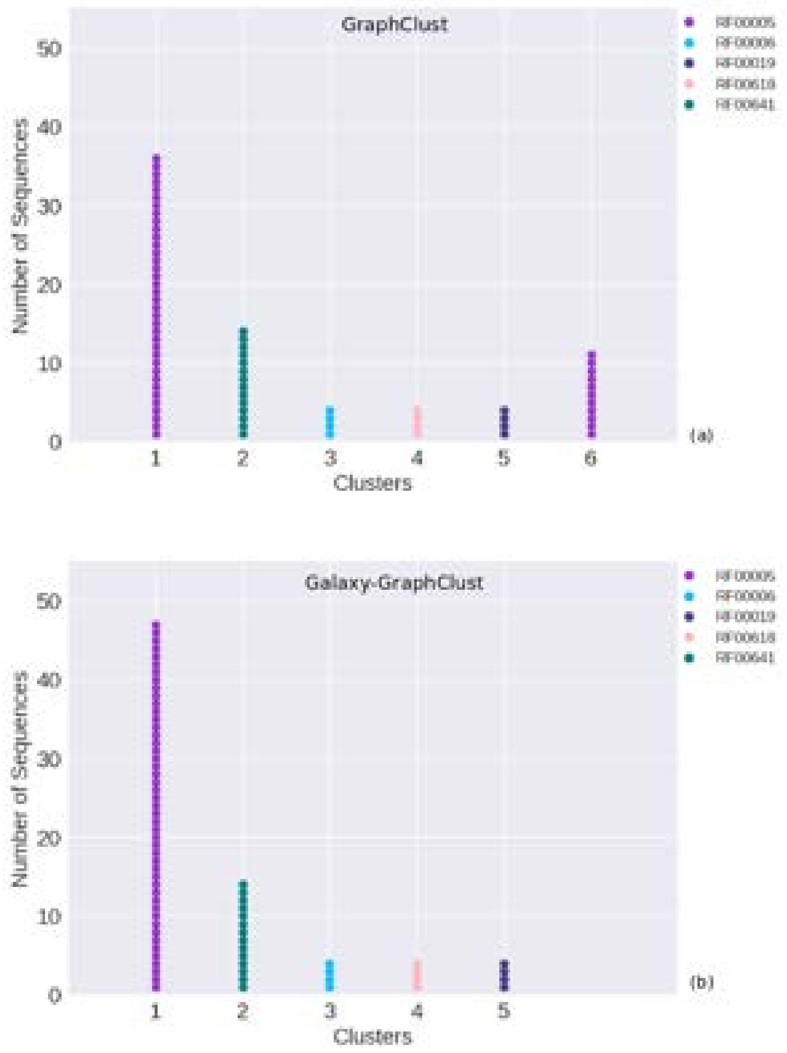
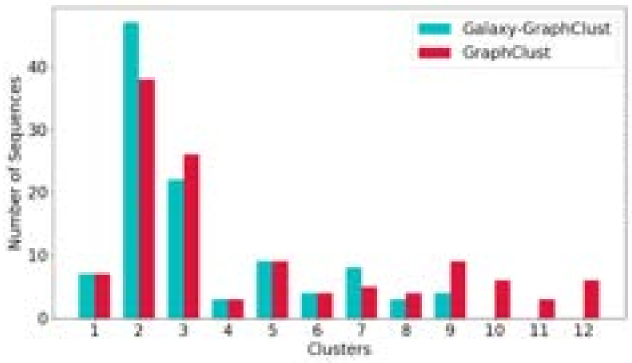


Figure 5.2: Rfam-cliques Low. Each color represents an RNA family. **(a)** Clusters obtained from GhaphClust. Cluster 1 and 6 contain members of the same family but were not merged together. **(b)** Clusters obtained from Galaxy-GhaphClust.

Method \ Metric	Completeness	Homogeneity	Adjusted Rand	V Measure
GraphClust	0.747	0.999	0.723	0.855
Galaxy-GraphClust	0.852	1.0	0.980	0.920

Table 5.2: Rfam-cliques Low. Evaluation of clusters using scikit-learn evaluation metrics. Galaxy-GraphClust outperforms the original pipeline according to these four metrics.

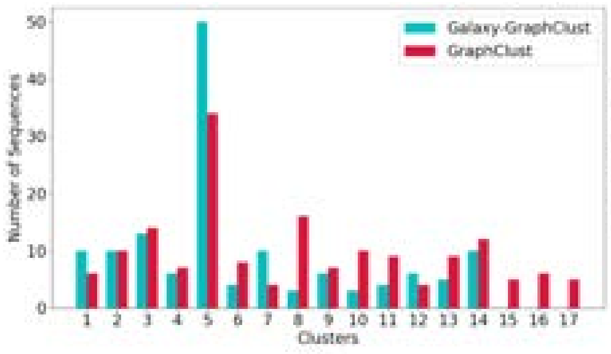


(a)

	GraphClust	Galaxy-GraphClust
Round 1	3	3
Round 2	5	5
Round 3	7	7
Round 4	7	8
Round 5	12	9

(b)

Figure 5.3: Rfam-cliques Medium. Summary of clusters after 5 **rounds**. GraphClust predicted 12 clusters with in total 120 sequences, and Galaxy-GraphClust clustered 107 sequences in 9 clusters. **(a)** The height of the bars represents the number of sequences in each cluster. **(b)** The number of predicted clusters after each round.



(a)

	GraphClust	Galaxy-GraphClust
Round 1	3	3
Round 2	8	7
Round 3	11	10
Round 4	14	13
Round 5	17	14

(b)

Figure 5.4: Rfam-cliques High. Summary of clusters after 5 **rounds**. GraphClust predicted 17 clusters with in total 163 sequences, and Galaxy-GraphClust clustered 140 sequences in 14 clusters. **(a)** The height of the bars represents the number of sequences in each cluster. **(b)** The number of predicted clusters after each round.

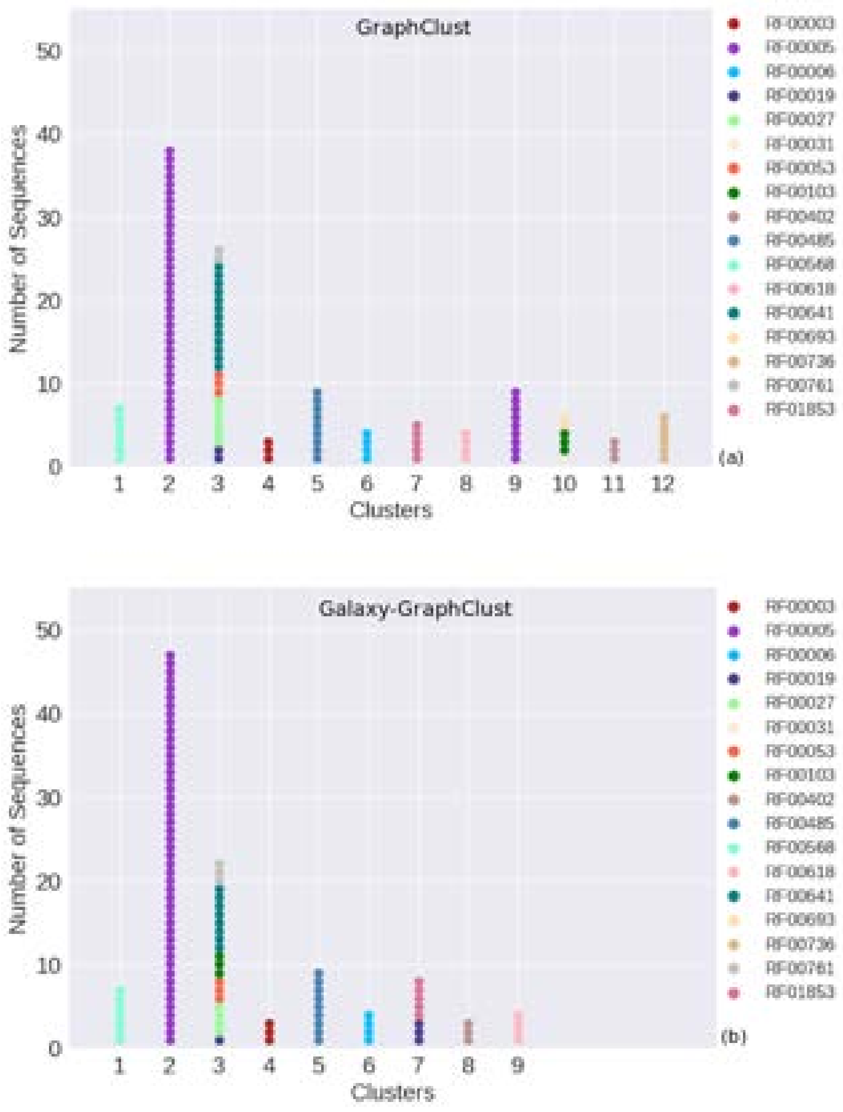


Figure 5.5: Rfam-cliques Medium Each color represents an RNA family. **(a)** Clusters obtained from GhaphClust. **(b)** Clusters obtained from Galaxy-GhaphClust.

Metric **Method**	Completeness	Homogeneity	Adjusted Rand	V Measure
GraphClust	0.812	0.913	0.703	0.859
Galaxy-GraphClust	0.801	0.911	0.847	0.852

Table 5.3: Rfam-cliques Medium. Evaluation of clusters using scikit-learn evaluation metrics. Even though GraphClust has slightly higher scores for completeness, homogeneity and v measure, the adjusted Rand score for Galaxy-GraphClust is higher by $\sim 14\%$ which states that the clusters predicted by Galaxy-GraphClust are less random than ones from the original GraphClust implementation.

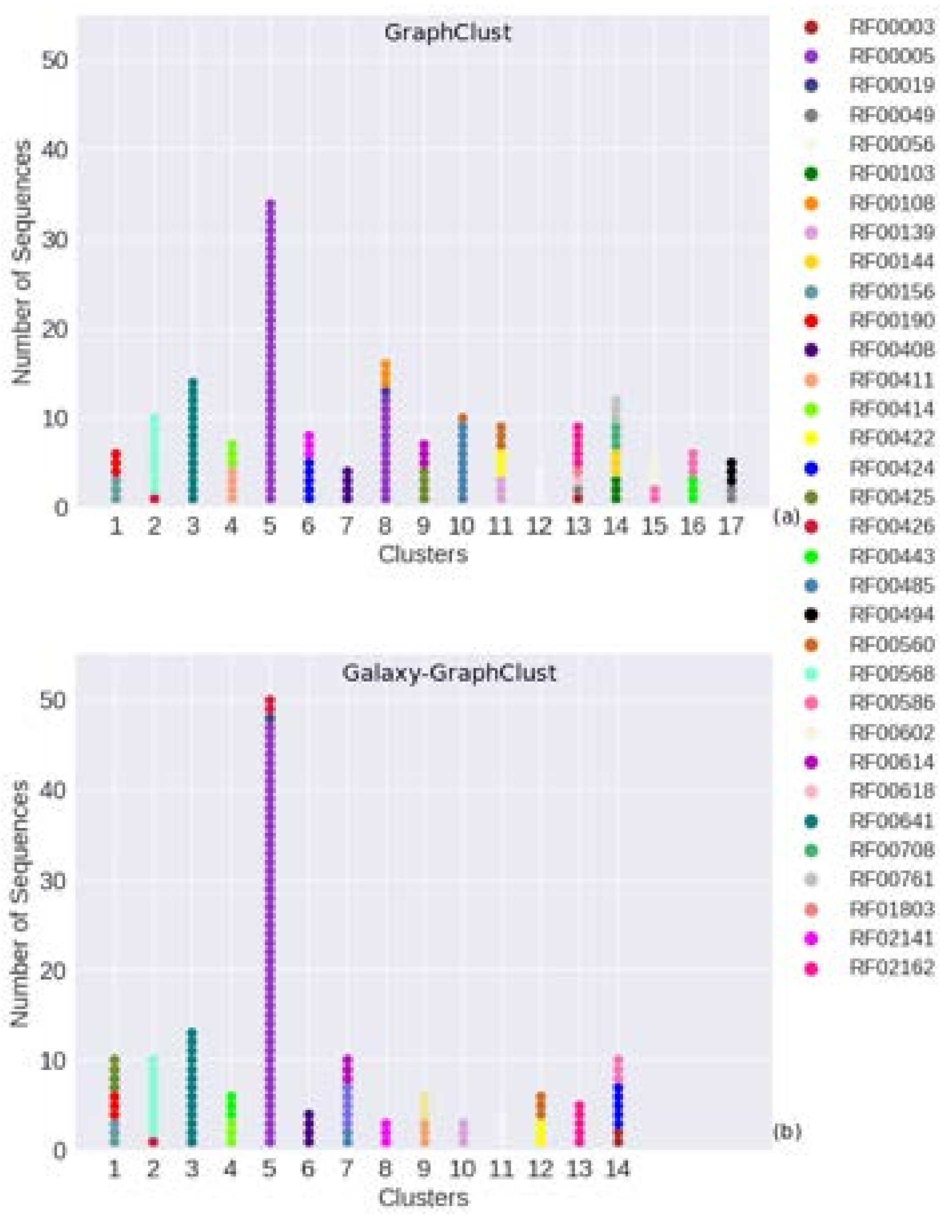


Figure 5.6: Rfam-cliques High Each color represents an RNA family. **(a)** Clusters obtained from GhaphClust. **(b)** Clusters obtained from Galaxy-GhaphClust.

Metric **Method**	Completeness	Homogeneity	Adjusted Rand	V Measure
GraphClust	0.840	0.896	0.666	0.867
Galaxy-GraphClust	0.845	0.926	0.841	0.883

Table 5.4: Rfam-cliques High. Evaluation of clusters using scikit-learn evaluation metrics. Galaxy-GraphClust outperforms the original pipeline according to all four metrics.

This benchmarking showed that, even though in some cases Galaxy-GraphClust clustered less sequences than GraphClust, it improved the quality of predicted clusters for all three datasets.

5.2 Exemplary Use Case

As a use case for the approach, we worked with a *metatranscriptome* dataset. A metatranscriptome is a transcriptome of a several interacting organisms or species. Transciptome is the entire RNA content of the cell [48], which is constructed during the process of transcription (Sec. 2.1.1). In other words, the metatranscriptome is a collection of entire RNAs from several organisms. In our case, the dataset we worked with was dominated by *cyanobacteria*, specifically *Trichodesmium*. Cyanobacteria are a large and diverse phylum in the kingdom Bacteria. It includes many different species. Cyanobacteria are *oxygenic phototrophs*, which means that they use photosynthesis to acquire energy from light, then they use that acquired energy to convert carbon dioxide from the air into the nutrients needed for growth. During the photosynthesis process oxygen is generated. This property makes cyanobacteria unique, because no other known bacteria can generate oxygen. Because they are bacteria, they are quite small and usually unicellular, however, they often grow in colonies large enough to be seen with bare eye [49]. Trichodesmium, also known as *sea sawdust* is a species of cyanobacteria. It is a colonial marine cyanobacteria. They can be found in tropical and subtropical ocean waters as well as in Red Sea. The distinguishing property of this bacteria is its ability to fix atmospheric nitrogen into ammonium in daylight under aerobic conditions without the use of heterocysts. So far, it is the only known bacteria with this property [50].

Originally the dataset contained ~ 3.5 million sequences. To remove highly redundant sequences from the dataset, we performed a CD-Hit [42] clustering with similarity threshold of 0.9. This resulted in ~ 900000 clusters, and thus ~ 900000 representative sequences.

We created three datasets by randomly sampling 1000, 5000, 10000, 20000, 50000 and 100000 sequences from the CD-Hit pre-processed dataset and performed 2 rounds of clustering for each of them. The parameters for Galaxy-GraphClust where based on the default parameters of the original GraphClust pipeline with few differences. In contrast to the default configuration, the parameters for the pre-processing step were chosen to avoid fragmenting the sequences. More precisely, we chose window size of 10000 and window shift of 100%. For the graph encoding step, the group size parameter was set to 200 to parallelize, and hence speed up the process. And in the final post-processing step all sequences with E-value bigger than 0.001 were discarded.

In the following sections we present the quantitative analysis of the clustering results of the metatranscriptome dataset.

5.2.1 Quantitative Analysis

We started clustering the datasets according to their sizes in ascending order. The Galaxy-GraphClust was performed on the Freiburg Galaxy server [51], in a real life scenario with queuing of the tasks, thus, we were unable to measure the precise run time. However, from the overall run time of the Galaxy-GraphClust on the server, we could see that the run time of the approach is linear on the number of sequences (Fig. 5.7). We measured the

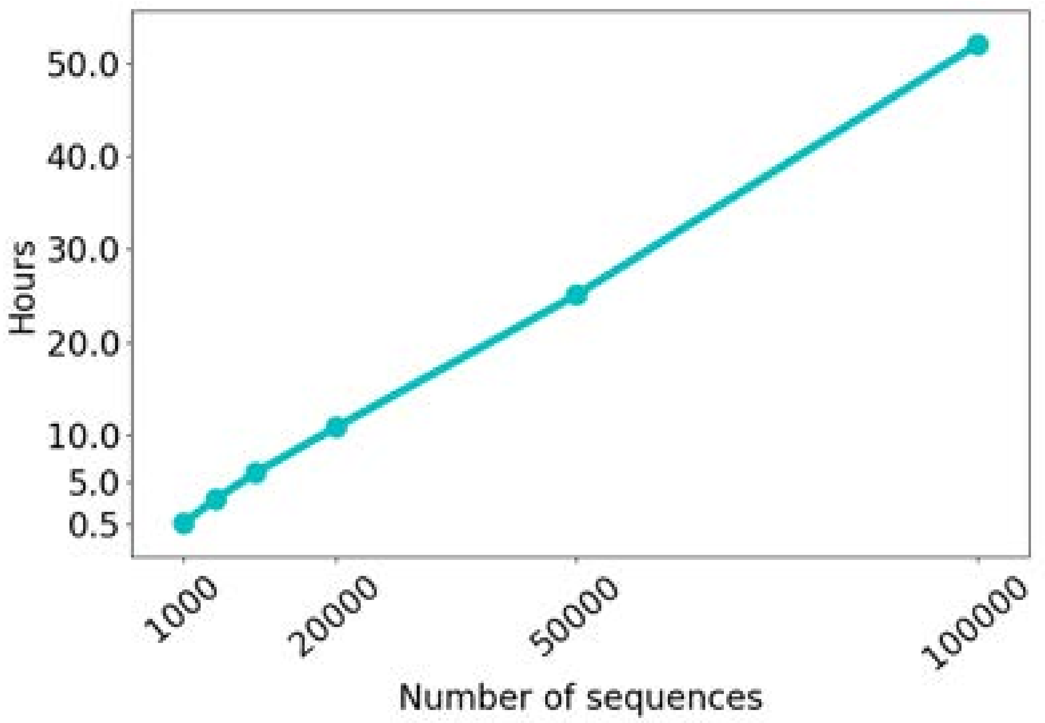


Figure 5.7: Run times. The run time of the Galaxy-GraphClust for 1 iteration on 6 datasets with 1000, 5000, 10000, 20000, 50000 and 100000 sequences. The run time is linear on the number of sequences.

run time of 1 iteration for each dataset. One iteration on the dataset of 1000 sequences was finished after approximately 30 minutes and the biggest dataset of 100000 sequences in about 50 hours.

We inspected the results of the datasets and here we present quantitative analysis of the results of two iterations for the dataset containing 100000 sequences. After the first iteration Galaxy-GraphClust predicted 77 clusters with 5085 sequences. However, the original number of predicted clusters was 89, but several clusters were merged together resulting in the final number. After the second round 95 new clusters were predicted but 3 of them where merged, resulting in total 169 clusters with 6777 sequences (Tab. 5.5). Fig. 5.8

	Before merge	**After merge**	**Number of sequences**
Round 1	89	77	5085
Round 2	95	92	1701
Final		169	6777

Table 5.5: Number of predicted clusters after each round. The last row represents the final number of clusters and clustered sequences.

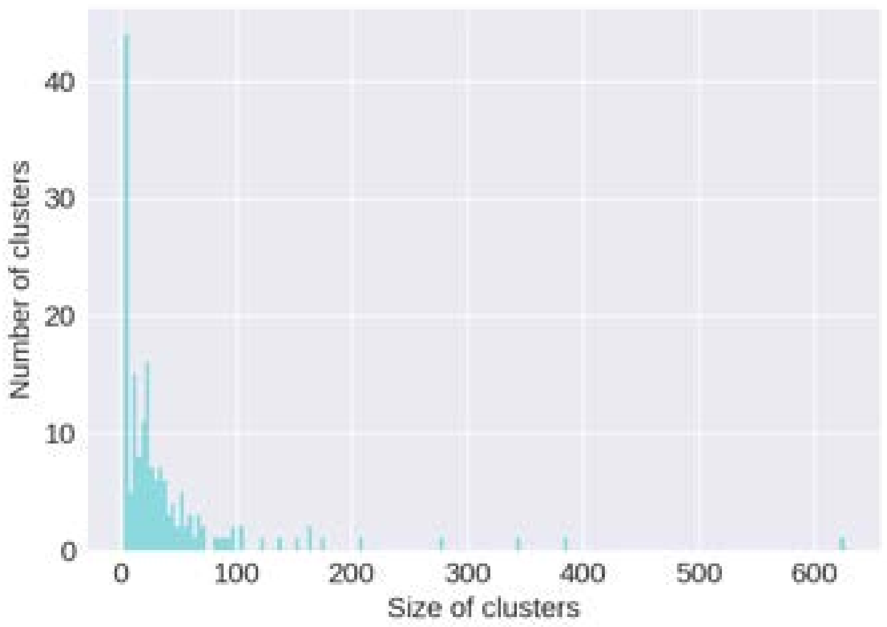


Figure 5.8: Clusters sizes. Distribution of the cluster sizes after the first round. The x-axis represent the size of each cluster, while y-axis represent number of clusters containing that number of sequences. Majority of the clusters contain few sequences, however 11 clusters contain more than 100 sequences.

shows the sizes of the final clusters. As it can be seen from the visualization most of the predicted cluster contain only few sequences. However, one of the predicted clusters contains more than 600 sequences. The size of the smallest allowed cluster can be set in the last step of the pipeline, and in our case we set it to 3.

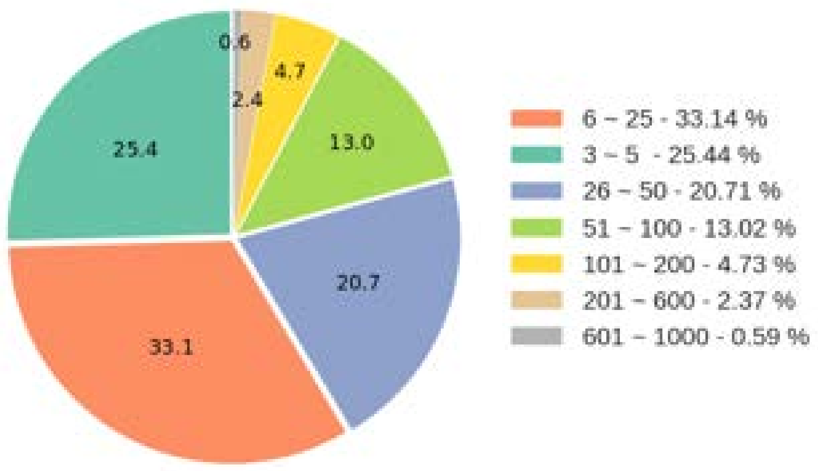


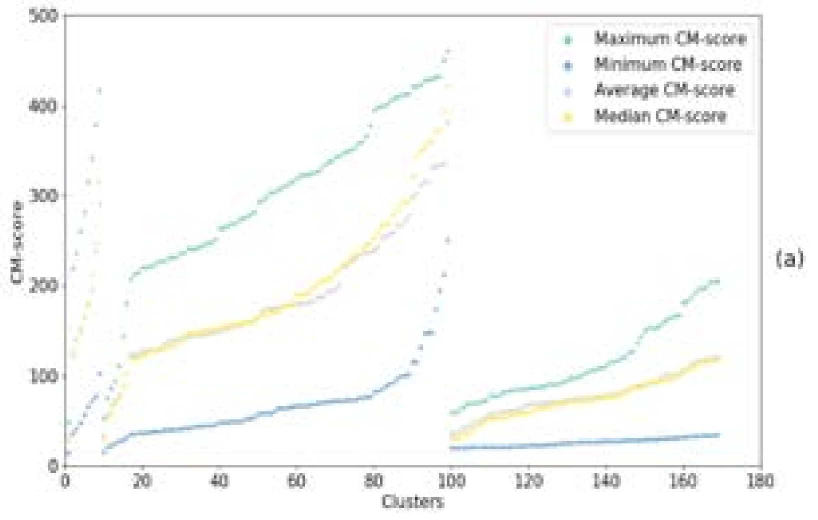
Figure 5.9: Ratio of clusters sizes. Each segment of the pie chart represents the ratio of clusters containing number of sequences within the indicated range, e.g. 13.0% of all the clusters contain from 51 to 100 sequences.

In Fig. 5.9 are shown the percentages of clusters according to their sizes i.e. each segment of the pie chart represents the ratio of clusters containing number of sequences within the indicated range, e.g. 33.1% of all the clusters contain from 6 to 25 sequences.

We also examined covariance model scores (Sec. 2.3.3) for the clustered sequences. The maximum, minimum, average and median values for CM scores for each cluster are shown in Fig. 5.10 (a). After the first iteration 77 clusters were predicted, and it can be seen from the figure that sequences clustered in the first iteration have higher CM scores than the majority of the sequences clustered in the second iteration. This makes sense because NSPDK finds candidate clusters from the most dense neighborhoods and that is why the sequences clustered in the first iteration have higher CM score than the ones clustered in the second iteration. The distribution of CM scores for all the clusters is shown via boxplots [52] in Fig. 5.10 (b). The horizontal axis is showing four numerical attributes listed below

- maximum CM scores
- minimum CM scores
- average CM scores
- median CM scores

On each box, the central mark is the median, the edges of the box are the lower hinge (defined as the 25th percentile) and the upper hinge (the 75th percentile), the whiskers extend to the most extreme data points not considered outliers,



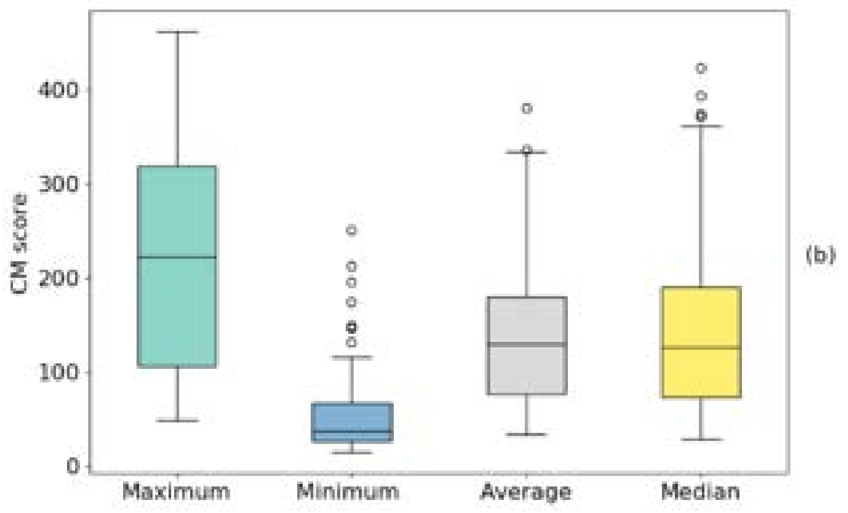


Figure 5.10: Destribution of CM scores. **(a)** The maximum, minimum, average and median CM scores per cluster. **(b)** The distribution of CM scores for all the clusters. Sequences predicted in the first iteration have on average higher scores than the onces predicted in the second iteration.

these ones are plotted individually as circles. Using the graph we can compare the range and distribution of the CM scores for all the clusters. We computed the listed values for each cluster. As a results we had four lists containing the maximum, minimum, average and median values for each cluster. As it can be seen in the Fig. 5.10 (b), the median of the highest CM scores is around ~ 220, the median of the lowest scores is ~ 35, the average is ~ 130 and the median CM score for all the sequences in all the clusters is ~ 125.

6 Conclusion and Future Perspectives

Invention of new technologies and fast growing computational power greatly increased the acquisition of new knowledge. The accumulation of scientific knowledge and new computational technologies has changed human life and our understanding of life. Today, it is barely possible to imagine any field of science that is not using computational methods, and especially the fields of life sciences. However, not every life scientist has programming or computer science skills, which yields to problems during installation and usage of essential tools and methods. Due to this, the accessibility problem of computational tools must be addressed, or in other words, a solution must be provided, which will enable scientists to easily integrate and run even complicated tools.

In this work we presented Galaxy-GraphClust, an upgraded and improved version of an iterative approach for alignment-free, structural clustering of RNA sequences called GraphClust. We split the original pipeline into pieces and created an independent module for each part, ensuring the modularity of the pipeline. Moreover, we integrated each tool in an open web-based platform, called Galaxy framework. The Galaxy framework contains numerous tools and enables users to perform computational analysis of genomic data. It is independent from the platform users are working on, which means anyone with access to the Internet can use it. Furthermore, Galaxy ensures accessibility of its tools and reproducibility of the analysis made in the framework. After integration of our modules in the Galaxy framework, we connected independent modules into a Galaxy workflow. Galaxy workflows allow users to run a set of tools at once. By using workflows we recreated the structure of the pipeline from single modules.

The modularity level of Galaxy-GraphClust allows to easily upgrade or exchange single modules of the pipeline, while original GraphClust, due to its monolithic configuration, strictly depends on all the third-party tools it is using, and thus, is hard to maintain. Furthermore, the integration into the Galaxy framework, allows users to extend the pipeline with any pre- or post-processing tools available, and provides the freedom to exchange the steps.

We compared Galaxy-GraphClust with GraphClust on several labeled datasets. Using both approaches, we performed five iterations on three small sets of RNA sequences from the Rfam-cliques dataset. We evaluated predicted clusters using various machine learning evaluation metrics. The evaluation showed that Galaxy-GraphClust outperforms the original GraphClust for all three cases.

Additionally, we tested Galaxy-GraphClust on a real world use case of a metatranscriptome dataset. We performed two clustering iterations on datasets of 1000, 5000, 10000, 20000, 50000 and 100000 sequences. These tests demonstrate that Galaxy-GraphClust can not only handle big datasets as well as the original pipeline, but also that it has linear run time complexity on the number of the sequences.

Despite these encouraging results, there is room for improvement in the future. It would be interesting to extend and exchange parts of the pipeline (e.g. integrate RNA structure probing experiments), in order to find better and more efficient combination of individual modules. Another direction of interesting research would be the improvement of the iterations in Galaxy. Currently Galaxy does not provide a feature to rerun an entire workflow from the history like it is possible to do with single tools, or to change parameters for subworkflows. However, Galaxy is a constantly developing and improving framework and hopefully in near future it will be possible to improve the iterative part of the Galaxy-GraphClust.

Clustering the entire metartancsriptome dataset presented earlier and gaining biologically meaningful results was beyond the scope of this work. However, it would be interesting to cluster the entire dataset in the proceeding works and for example compare the predicted clusters with existing genome databases in order to find possible matches.

Bibliography

[1] Steffen Heyne, Fabrizio Costa, Dominic Rose, and Rolf Backofen. Graphclust: alignment-free structural clustering of local rna secondary structures. *Bioinformatics*, 28(12):i224–i232, 2012.

[2] Galaxy: an open platform for accessible, reproducible, and transparent biomedical research. `http://galaxyproject.org`.

[3] Francis Crick et al. Central dogma of molecular biology. *Nature*, 227(5258):561–563, 1970.

[4] Science explained. `http://science-explained.com/theory/dna-rna-and-protein/`.

[5] N. B. Leontis. The non-watson-crick base pairs and their associated isostericity matrices. *Nucleic Acids Research*, 30(16):3497–3531, aug 2002.

[6] P. Clote and R. Backofen. *Computational Molecular Biology: An Introduction*. Wiley Series in Mathematical & Computational Biology. Wiley, 2000.

[7] Unravelling the double helix. `http://www.yourgenome.org/stories/unravelling-the-double-helix`.

[8] Ehud Lamm and Ron Unger. *Biological computation*. CRC Press, 2011.

[9] Suzanne Clancy and William Brown. Translation: Dna to mrna to protein. *Nature Education*, 1(1):101, 2008.

[10] Harvey Lodish, Arnold Berk, S Lawrence Zipursky, Paul Matsudaira, David Baltimore, and James Darnell. Molecular cell biology: An integrated view of cells at work. 2000.

[11] The complexity of living systems. `https://complexityoflivingcell.wordpress.com/tag/codon/`.

[12] Peter Steffen and Robert Giegerich. *BMC Bioinformatics*, 6(1):224, 2005.

[13] Joel L Sussman, Stephen R Holbrook, R Wade Warrant, George M Church, and Sung-Hou Kim. Crystal structure of yeast phenylalanine transfer rna: I. crystallographic refinement. *Journal of molecular biology*, 123(4):607–630, 1978.

[14] Marat M Yusupov, Gulnara Zh Yusupova, Albion Baucom, Kate Lieberman, Thomas N Earnest, JHD Cate, and Harry F Noller. Crystal structure of the ribosome at 5.5 å resolution. *science*, 292(5518):883–896, 2001.

[15] Robert W Holley, Jean Apgar, George A Everett, James T Madison, Mark Marquisee, Susan H Merrill, John Robert Penswick, and Ada Zamir. Structure of a ribonucleic acid. *Science*, 147(3664):1462–1465, 1965.

[16] J. S. Mattick. Non-coding RNA. *Human Molecular Genetics*, 15(90001):R17–R29, apr 2006.

[17] Rolf Backofen, Stephan H. Bernhart, Christoph Flamm, Claudia Fried, Guido Fritzsch, Jörg Hackermüller, Jana Hertel, Ivo L. Hofacker, Kristin Missal, Axel Mosig, Sonja J. Prohaska, Dominic Rose, Peter F. Stadler, Andrea Tanzer, Stefan Washietl, and Sebastian Will. Rnas everywhere: genome-wide annotation of structured rnas. *Journal of Experimental Zoology Part B: Molecular and Developmental Evolution*, 308B(1):1–25, 2007.

[18] Rna bioinformatics lecture, uni freiburg. `www.bioinf.uni-freiburg.de`.

[19] Philippe Brion and Eric Westhof. HIERARCHY AND DYNAMICS OF RNA FOLDING. *Annual Review of Biophysics and Biomolecular Structure*, 26(1):113–137, jun 1997.

[20] Michael Zuker and David Sankoff. Rna secondary structures and their prediction. *Bulletin of mathematical biology*, 46(4):591–621, 1984.

[21] Paul P Gardner and Robert Giegerich. A comprehensive comparison of comparative rna structure prediction approaches. *BMC Bioinformatics*, 5(1):140, 2004.

[22] Stephan H Bernhart, Ivo L Hofacker, Sebastian Will, Andreas R Gruber, and Peter F Stadler. RNAalifold: improved consensus structure prediction for RNA alignments. *BMC Bioinformatics*, 9(1):474, 2008.

[23] David Sankoff. Simultaneous solution of the rna folding, alignment and protosequence problems. *SIAM Journal on Applied Mathematics*, 45(5):810–825, 1985.

[24] S. Will, T. Joshi, I. L. Hofacker, P. F. Stadler, and R. Backofen. LocARNA-p: Accurate boundary prediction and improved detection of structural RNAs. *RNA*, 18(5):900–914, mar 2012.

[25] Bruce A Shapiro and Kaizhong Zhang. Comparing multiple rna secondary structures using tree comparisons. *Computer applications in the biosciences: CABIOS*, 6(4):309–318, 1990.

[26] Kaizhong Zhang and Dennis Shasha. Simple fast algorithms for the editing distance between trees and related problems. *SIAM journal on*

computing, 18(6):1245–1262, 1989.

[27] Christina Otto, Mathias Möhl, Steffen Heyne, Mika Amit, Gad M Landau, Rolf Backofen, and Sebastian Will. ExpaRNA-p: simultaneous exact pattern matching and folding of RNAs. *BMC Bioinformatics*, 15(1), dec 2014.

[28] Sebastian Will, Christina Otto, Milad Miladi, Mathias Möhl, and Rolf Backofen. SPARSE: quadratic time simultaneous alignment and folding of RNAs without sequence-based heuristics. *Bioinformatics*, 31(15):2489–2496, apr 2015.

[29] Eric P Nawrocki and Sean R Eddy. Infernal 1.1: 100-fold faster rna homology searches. *Bioinformatics*, 29(22):2933–2935, 2013.

[30] Karim Lari and Steve J Young. The estimation of stochastic context-free grammars using the inside-outside algorithm. *Computer speech & language*, 4(1):35–56, 1990.

[31] David Haussler. Convolution kernels on discrete structures. Technical report, Citeseer, 1999.

[32] Fabrizio Costa and Kurt De Grave. Fast neighborhood subgraph pairwise distance kernel. In *Proceedings of the 26th International Conference on Machine Learning*, pages 255–262. Omnipress, 2010.

[33] Andrei Z Broder. On the resemblance and containment of documents. In *Compression and Complexity of Sequences 1997. Proceedings*, pages 21–29. IEEE, 1997.

[34] Enis Afgan, Dannon Baker, Marius van den Beek, Daniel Blankenberg, Dave Bouvier, Martin Čech, John Chilton, Dave Clements, Nate Coraor, Carl Eberhard, Björn Grüning, Aysam Guerler, Jennifer Hillman-Jackson, Greg Von Kuster, Eric Rasche, Nicola Soranzo, Nitesh Turaga, James Taylor, Anton Nekrutenko, and Jeremy Goecks. The galaxy platform for accessible, reproducible and collaborative biomedical analyses: 2016 update. *Nucleic Acids Research*, 44(W1):W3–W10, may 2016.

[35] Public galaxy service. `http://usegalaxy.org`.

[36] Galaxy add tool tutorial. `https://wiki.galaxyproject.org/Admin/Tools/AddToolTutorial`.

[37] Jeremy Goecks, Anton Nekrutenko, James Taylor, and The Galaxy Team. Galaxy: a comprehensive approach for supporting accessible, reproducible, and transparent computational research in the life sciences. *Genome Biology*, 11(8):R86, 2010.

[38] Stephen F Altschul, Thomas L Madden, Alejandro A Schäffer, Jinghui Zhang, Zheng Zhang, Webb Miller, and David J Lipman. Gapped blast

and psi-blast: a new generation of protein database search programs. *Nucleic acids research*, 25(17):3389–3402, 1997.

[39] Peter Steffen, Björn Voß, Marc Rehmsmeier, Jens Reeder, and Robert Giegerich. Rnashapes: an integrated rna analysis package based on abstract shapes. *Bioinformatics*, 22(4):500, 2005.

[40] Intro to conda. `https://conda.io/docs/intro.html`.

[41] S. Janssen and R. Giegerich. The RNA shapes studio. *Bioinformatics*, 31(3):423–425, oct 2014.

[42] Weizhong Li and Adam Godzik. Cd-hit: a fast program for clustering and comparing large sets of protein or nucleotide sequences. *Bioinformatics*, 22(13):1658–1659, 2006.

[43] Elena Rivas, Jody Clements, and Sean R Eddy. A statistical test for conserved RNA structure shows lack of evidence for structure in lncRNAs. *Nature Methods*, 14(1):45–48, nov 2016.

[44] Sam Griffiths-Jones, Alex Bateman, Mhairi Marshall, Ajay Khanna, and Sean R Eddy. Rfam: an rna family database. *Nucleic acids research*, 31(1):439–441, 2003.

[45] Milad Miladi, Alexander Junge, Fabrizio Costa, Stefan E Seemann, Jakob Hull Havgaard, Jan Gorodkin, and Rolf Backofen. Rnascclust: clustering rna sequences using structure conservation and graph based motifs. *Bioinformatics*.

[46] 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, et al. Rfam 12.0: updates to the rna families database. *Nucleic acids research*, page gku1063, 2014.

[47] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[48] Terence A Brown. *Genomes*. Garland science, 2006.

[49] Life history and ecology of cyanobacteria. University of California Museum of Paleontology. Retrieved 17 July 2012. `http://www.ucmp.berkeley.edu/bacteria/cyanointro.html`.

[50] Edward J Carpenter and Douglas G Capone. *Marine pelagic cyanobacteria: Trichodesmium and other diazotrophs*, volume 362. Springer Science & Business Media, 2013.

[51] The Freiburg Galaxy Project. `http://galaxy.uni-freiburg.de/`.

[52] Robert Mcgill, John W. Tukey, and Wayne A. Larsen. Variations of box plots. *The American Statistician*, 32(1):12–16, 1978.