*CPSC 290: Directed Research in Computer Science, Spring 2023, Yale University*

# Novel Pipeline for RNA Discovery & Analysis Applied to Sodium-Binding Riboswitches

By Julian Sanker, advised by Dr. Smita Krishnaswamy, under the mentorship of Dr. Ronald Breaker and Christopher King. Link to GitHub repository.

## Table of Contents

## Abstract

This project investigates the construction of a Python framework for RNA discovery and analysis, with the goal of improving the computational tools available to RNA researchers. The framework consists of three parts: 1) a pipeline for refinement and analysis of RNA models, 2) a genetic algorithm for RNA folding, and 3) a frontend website that interfaces with 1 and 2. The pipeline incorporates several bioinformatic software packages, including Infernal, HMMER, CMfinder, and R2R, to perform useful operations on RNA sequence data. Functions include: running a genome search, parsing the search results into a SQL database, filtering the data for use in generating new RNA models, and visualizing the results of a search. The genetic algorithm, incomplete, is based on the NeuroEvolution of Augmenting Topologies algorithm, adapted to evolve context-free grammars representing RNA models, and it is meant to replace the CMfinder step of the pipeline to predict RNA secondary structure/folding. The frontend website, also incomplete, presents a

unified interface for RNA researchers to run the various operations of the pipeline and view results, as well as a new RALEE-esque RNA alignment editor, to conduct experiments on sequence data without needing technical knowledge of the command-line or the idiosyncrasies of particular software packages. The pipeline will utilize a SQL database to store and manipulate data, and the website will be hosted on a remote server to allow for easy access by researchers. The pipeline was applied to analyze two sodium-binding riboswitch RNAs, nhaA-I and DUF1646, and shows promise for incorporating these models into a single, larger sodium-binding riboswitch model. More research is needed, so the experiment will be continued over the summer. A new framework for various computational RNA tasks will be a valuable tool for RNA researchers and a useful resource for the Breaker Lab in particular.

The final report can be found in the link to Spring 2023 Final Report.

## Introduction

*Project developed during Spring 2023 for CPSC 290: Directed Research in Computer Science at Yale University, under the mentorship of Dr. Ronald Breaker and Christopher King.*

The main goal of the project is to improve computational tools for structured RNA discovery and analysis. The project is divided into three parts: the first is the development of a pipeline that interfaces with numerous bioinformatic software packages, including Infernal, HMMER, CMfinder, and R2R, to perform useful operations on RNA sequence data across these tools. The second is the incorporation of a genetic algorithm into the pipeline, in place of the existing, outdated CMfinder software, to predict RNA secondary structure/folding. The final component is a frontend website that provides an interface for RNA researchers to utilize the pipeline and its various operations independently, as well as a new RALEE-esque RNA alignment editor, to conduct experiments on sequence data without needing technical knowledge of the particular software packages.

The project is incomplete, to be continued over the summer. The following sections describe the project in more detail. The project proposal from the beginning of the semester can be found in the link to Spring 2023 Project Proposal.

## Project Components

The project is divided into three components: a backend pipeline, a genetic algorithm, and a frontend website. The backend pipeline is currently being redesigned, and the genetic algorithm and frontend website are currently incomplete. The following sections describe each component in more detail.

### 1. Backend Pipeline

A backend pipeline that provides functions to run a genome search on a given RNA model, parse the search results into a SQL database, and then filter the data for use in generating new RNA models. The pipeline is written in Python and utilizes the Infernal, HMMER, CMfinder, and R2R software packages. The pipeline is designed to be modular, so that it can be easily extended to incorporate new software packages and operations. The pipeline is also designed to be run on a remote server, so that it can be accessed by the frontend website. The main workhorse of the pipeline is the Infernal software package, which uses stochastic context-free grammars as representations of RNA secondary structure. The pipeline is currently incomplete, but the following functions have been implemented:

- `cmbuild_submit`: Starts Slurm job to build and calibrate a covariance model for a given RNA sequence alignment using Infernal.
- `cmsearch_submit`: Starts Slurm job to search a given RNA model against a given genome database using Infernal's CMsearch.
- `cmsearch_parse`: Parses the results of a search into a SQLite database.
- `cmsearch_analyze`: Analyzes the results of a search by plotting the E-value score distribution of the hits, filtering the hits by E-value score and uniqueness and reporting count numbers, and running R2R on the hits to visualize the resulting model. E-value score is a measure of the statistical significance of a hit. For example, an E-value of 1 indicates an expectation of ~1 false positive at that rank.
- `runr2r`: Runs R2R on a given RNA model to visualize the results of a CMsearch search.
- `sto_reformat`: Reformats a given RNA sequence alignment from Stockholm to FASTA (unannotated) format.
- `run_cmfinder`: Starts a Slurm job to run CMfinder on a given FASTA sequence file to predict RNA secondary structure/folding. CMfinder is a learning step to infer a stochastic context-free grammar, representing an RNA secondary structure, from a set of sequences.

The pipeline is currently being redesigned to utilize a SQL database instead of a filesystem + in-memory data structures to store and manipulate data. The pipeline is also being redesigned to be more modular, so that it can be easily extended to incorporate new software packages and operations. However, this means that some functionality was reversed in the process of redesigning the pipeline, and some functionality is currently missing. The following functions are currently missing:

- `cmsearch_compare`: Compares the results of two searches to determine the number of hits that are unique to each search, and the number of hits that are shared between them.

In lieu of a frontend website, the pipeline is currently accessed through a command-line interface (CLI) that provides commands for running each of the pipeline functions, or the pipeline can be run directly from the Python interpreter. The CLI is implemented using the Python Click library. It is possible to start a process that runs the entire pipeline in the background on the cluster, and then check the status of the process later.

### 2. Genetic Algorithm

A genetic algorithm that predicts RNA secondary structure/folding for a given RNA sequence alignment. The genetic algorithm is written in Python, and is meant to replace the CMfinder step of the pipeline, since CMfinder is out-of-date and uses the Perl programming language. The genetic algorithm is currently incomplete, but a prototype has been implemented, nick-named TrEAT, or Tree Evolution of Augmenting Topologies, which refers to the usage of a tree data structure to represent the stochastic context-free grammar of the RNA secondary structure/folding, and is strongly based on the famous Neuro-Evolution of Augmenting Topologies (NEAT) algorithm for evolving neural networks. The following functionality has been implemented:

- `Rule`: A class that represents a rule in the context-free grammar of the RNA secondary structure/folding. Each rule consists of a sequence that describes the primary structure of the RNA, with id's that refer to other rules in the grammar. Each rule should also have a probability of being chosen, but this functionality has not yet been implemented. The rule class also contains functions to mutate the rule, to generate a random rule, and to crossover two rules

- `Genome`: A class that represents a genome, which is a collection of 'rule' genes that encode a context-free grammer. The genome class also contains functions to mutate the genome, generate a random genome, and crossover two genomes. The crossover function is designed as in NEAT, using a historical marking scheme to align the genes of the two genomes to solve the competing conventions problem.
- `Organism`: A class that represents an organism, which is a genome that has a fitness score and belongs to a species.
- `Species`: A class that represents a species, which is a collection of organisms with similar genomes, and is used to maintain diversity in the population. Originally devised by Kenneth Stanley in the NEAT algorithm to solve the problem of protecting structural innovation in the population.
- `Population`: A class that represents a population, which is a collection of organisms and their species. The population class also contains functions to evolve the population, which is done by speciating the organisms, killing off the worst organisms in each species, and then breeding the remaining organisms in each species to generate new organisms.
- `Simulation`: A class that represents a simulation, which has a population and defines the operations of reproduction/evolution for the population. The simulation class also contains functions to run the simulation, which is done by running the reproduction/evolutionary operations for a given number of generations.

Once completed, the genetic algorithm must be adapted to fit into the pipeline.

### 3. Frontend Website

A frontend website that provides an interface for RNA researchers to utilize the pipeline and its various operations independently, as well as a new RALEE-esque RNA alignment editor, to conduct experiments on sequence data without needing technical knowledge of the particular software packages. The frontend is incomplete, and is currently implemented using the Python Flask framework and HTML/CSS/JavaScript. Ideally, the frontend will be redesigned to utilize the React.js library. The following functionality has been implemented:

- User authentication and authorization using the Flask-Login library.
- Admin password protection, since we plan to host the website on a public server.
- A prototype of the RALEE-esque RNA alignment editor, which is currently implemented using a custom Stockholm file parser plus JavaScript to render the alignment. The prototype is incomplete, and does not allow for editing or saving the alignment. We plan to reprogram the Stockholm alignment parser to utilize a numpy array to represent the alignment, making column/row operations faster, and to utilize the React.js library to render the alignment for editing.

Once the pipeline is completed, the frontend will be developed to enable users to 1) run pipeline operations step-by-step (a la carte?), and 2) run the entire pipeline, and to visualize the results of the operations. As of now, the frontend is not connected to the backend pipeline, and is only a prototype.

## Methodology

Much deliberation was put into the design of the pipeline and the project as a whole. Some of the design decisions are described below – although these are subject to change as the project progresses.

### Code Structure

The code for the pipeline is organized into the following directories:

- `data`: Contains the input and output data files for the pipeline, such as the RNA sequence alignments the CMs, R2R diagrams, and charts, separated into subdirectories for each step of the pipeline and for each RNA motif.
- `web`: Contains the code for the frontend website, separated into subdirectories for Flask blueprints, Jinja HTML templates, and JavaScript/CSS files.
- `treat`: Contains the code for the genetic algorithm.
- `cgk`: Contains some code files from Chris's project, which are currently unused but have been kept for reference. Methodology from these files has been incorporated into the pipeline.
- `jps`: Contains the pipeline code, separated into utility and file I/O functions, database models, pipeline operations, and bash scripts for running certain software packages.
- `cpsc290`: Contains the end-of-term report for the project – an abstract, a set of web pages describing the project, and a final project report in PDF format, which is a copy of this README file.

### Database Architecture

The pipeline uses a SQLite database to store the results of each step of the pipeline. The ORM (Object-Relational Mapping) library SQLAlchemy is used to interact with the database, which is organized into the following tables:

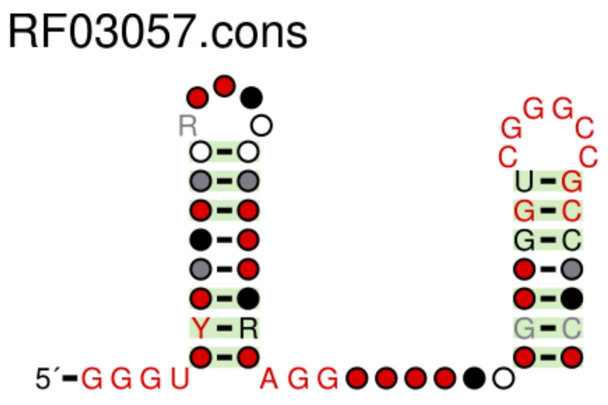
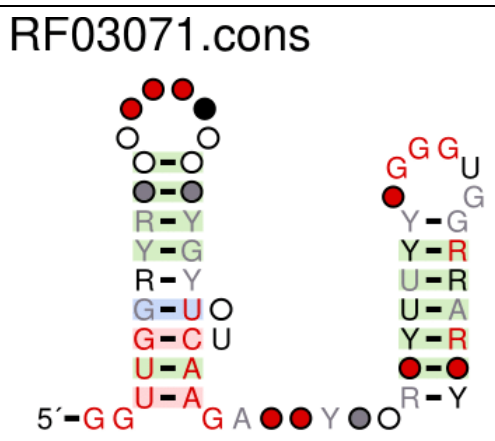
- `SlurmJob`: Represents a Slurm job, including the job ID, arguments passed, and the job status. The SlurmJob table is used to keep track of the jobs that are running on the cluster, and to check their status.
- `Search`: Represents a CMsearch search, including the search parameters (cmfile, target database, and other command-line flags), a timestamp, the Slurm job ID associated with the search, and associated hits found by the search. The Search table is used to keep track of the CMsearch searches that have been run, and to store the results of the searches.
- `Hit`: Represents a hit found by a CMsearch search, including the search that found the hit, the target sequence/chromosome accession identifier, the start and end positions of the hit in the target sequence, the strand (3' to 5' or 5' to 3'), the E-value score of the hit, and the alignment of the hit, as well as a few other fields from the CMsearch output. The Hit table is used to keep track of and store the hits that have been found by CMsearch searches.
- `Alnseq`: Represents an alignment of a hit, including a reference to the multiple sequence alignment containing the hit and the actual text of the aligned sequence. The Alnseq table is used to keep track of and store the alignments of hits.
- `StoFeature`: Represents a feature of a Stockholm alignment, including a reference to the Alnseq, the feature format (GC, GF, GS, or GR), field (e.g. SS_cons, seq, or RF), and text. The StoFeature table is used to keep track of and store the features of Stockholm alignments so that important annotations can be persisted when hits are filtered and written to new Stockholm alignment files.

## Experimental Process for Sodium-Binding Riboswitches

I am conducting an experiment using my pipeline to analyze two sodium-binding RNA motifs, known as nhaA-I and DUF1646, to determine whether they can be combined into a single representative model. My experimental process is as follows:

### 1. Download motifs

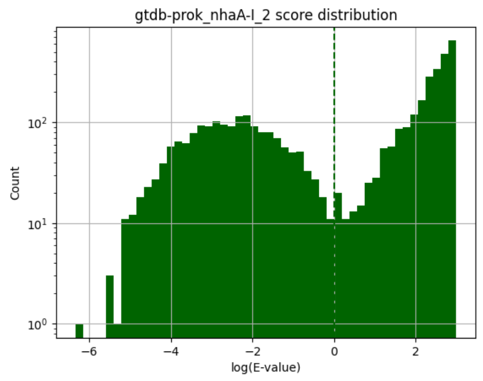
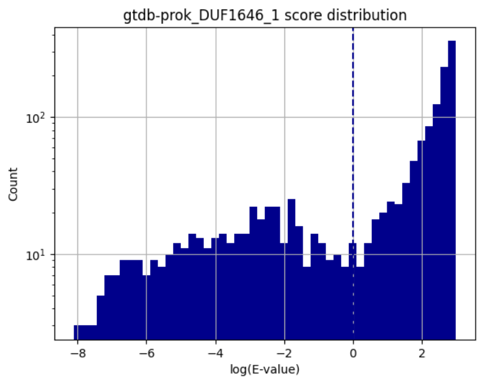
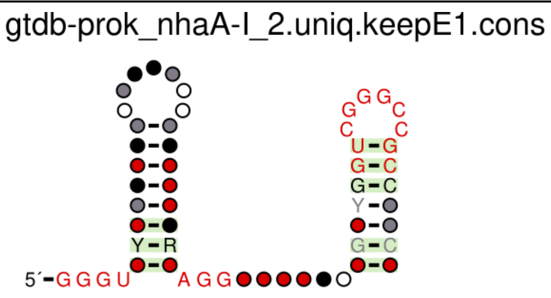
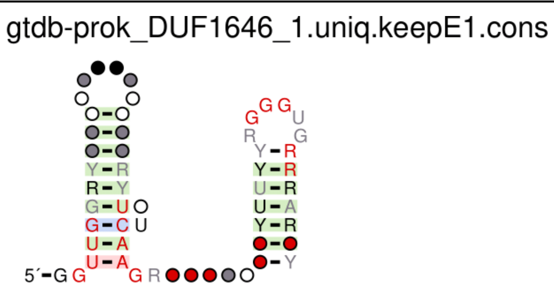
I downloaded the nhaA-I (RF03057) and DUF1646 (RF03071) RNA sequence alignments from the Rfam website. Shown below are R2R diagrams of the two alignments:

| | **nhaA-I** | **DUF1646** |
|---|---|---|
| **R2R diagram** |  |  |

As shown by the diagrams, the two RNAs are very similar, each with two stem-loops. The two RNAs also have similar lengths, with nhaA-I being 95 nucleotides long, and DUF1646 being 77 nucleotides long.

## 2. Search against genome database

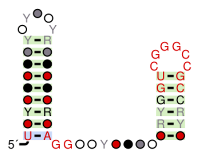
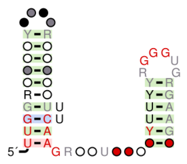
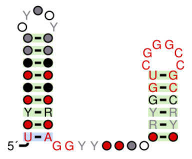
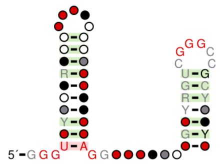
Using my pipeline, I ran CMsearch on the two alignments against a representative sample of bacterial genomes from Genome Taxonomy Database Release 207. The analyses of the results of the CMsearch searches, filtered by uniqueness and E-value threshold, are shown below:

| | **nhaA-I** | **DUF1646** |
|---|---|---|
| **Score distribution** |  |  |
| **R2R diagram** |  |  |
| **# hits** | 4365 | 1649 |
| **# unique hits** | 4091 | 1478 |

| | nhaA-I | DUF1646 |
|---|---|---|
| **# hits with E-value < 1** | 1796 | 497 |
| **# unique hits with E-value < 1** | 1663 | 421 |

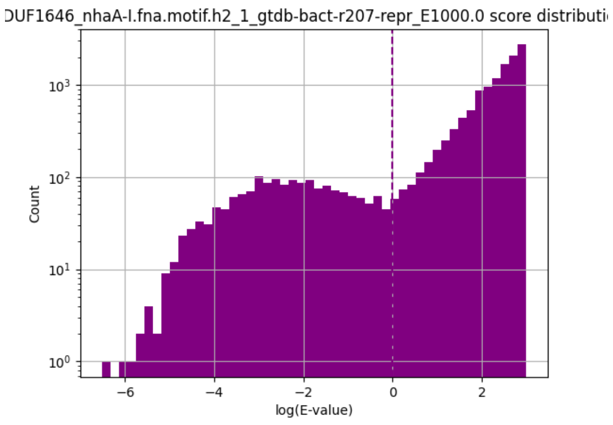
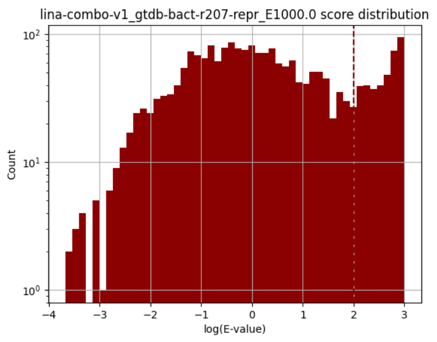
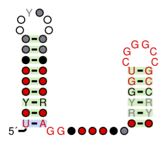
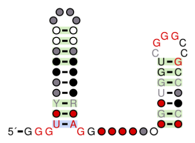
## 3. Refold using CMfinder

I reformatted the search results alignments (Stockholm files) as FASTA sequence files, and ran CMfinder on each of 1) the nhaA-I search results alignment, 2) the DUF1646 search results alignment, and 3) the combined search results alignment generated by concatenating the two FASTA files. The CMfinder results are shown below. Chris created another combined alignment, dubbed LiNa, using `esl-alimerge` to combine the original nhaA-I and DUF1646 Rfam alignment files, also shown below.

| | nhaA-I | DUF1646 | Combined | LiNa |
|---|---|---|---|---|
| **R2R diagram** | nhaA-I.fna.motif.h2_1.cons  | DUF1646.fna.motif.h2_1.cons  | DUF1646_nhaA-I.fna.motif.h2_1.cons  | lina-combo-v1.cons  |

## 4. Search new models against genome database

Then, to evaluate our combined models, I ran Infernal's `cmsearch` once more on the combined alignment generated by CMfinder, and on the LiNa alignment. The results of the searches are shown below:

| | Combined | LiNa |
|---|---|---|
| **Score distribution** |  |  |
| **R2R diagram** | DUF1646_nhaA-I.fna.motif.h2_1_gtdb-bact-r207-repr_E1000.0.uniq.keepE1.cons  | lina-combo-v1_gtdb-bact-r207-repr_E1000.0.uniq.keepE100.cons  |
| **# hits** | 14536 | 2354 |

| | Combined | LiNa |
|---|---|---|
| **# unique hits** | 13398 | 2183 |
| **# hits with E-value < 1** | 1818 | 1953 |
| **# unique hits with E-value < 1** | 1665 | 1801 |

As shown by the table, the combined model found far more hits than the LiNa model, but the LiNa model found more hits that were well-matched to the model (hits with E-value < 1).

### 5. Evaluate search results

Finally, I performed a few comparisons between the four searches, since we want to see how the combined models compare to the individual models.

First, we report hit counts for each search, including the total number of hits found, the number of unique hits found, and the number of unique hits found with an E-value less than 1000.0.

| Search Name | # Total | # Unique | # Unique E<1000 |
|---|---|---|---|
| gtdb-prok_DUF1646_1 | 1649 | 1478 | 1474 |
| gtdb-prok_nhaA-I_2 | 4365 | 4091 | 4080 |
| DUF1646_nhaA-I.fna.motif.h2_1_gtdb-bact-r207-repr_E1000.0 | 14536 | 13398 | 13355 |
| lina-combo-v1_gtdb-bact-r207-repr_E1000.0 | 2354 | 2183 | 2181 |

Then, we check for intersecting hits between the original nhaA-I and DUF1646 searches, finding zero intersecting hits:

```
Found 0 intersecting hits between DUF1646 and nhaA-I
```

Then, for each combined model, we compare the results of the combined model to the nhaA-I and DUF1646 searches to identify true positives (TP), false positives (FP), and false negatives (FN) by the following template:

| E | I | II | III | IV | V |
|---|---|---|---|---|---|
| 1 | ? | ? | ? | ? | ? |
| 10 | ? | ? | ? | ? | ? |

| E | I | II | III | IV | V |
|---|---|---|---|---|---|
| 100 | ? | ? | ? | ? | ? |
| 1000 | ? | ? | ? | ? | ? |

| Column | Description | Meaning |
|---|---|---|
| E | E-value threshold | |
| I | # of DUF1646 hits not in combo | FN |
| II | # of DUF1646 hits in combo | TP |
| III | # of combo hits not in DUF1646 or nhaA-I | FP* |
| IV | # of nhaA-I hits in combo | TP |
| V | # of nhaA-I hits not in combo | FN |

*Some of the false positives in III may actually be true positives. For example, if a hit is found in the combined model, but not in either of the individual models, it could be a true positive that was missed by the individual models, or it could be a false positive that was found by the combined model – we can't tell without further analysis.

The results of the comparisons of the combined models to the original models are shown below:

**DUF1646_nhaA-I vs. DUF1646 and nhaA-I**

| E | I | II | III | IV | V |
|---|---|---|---|---|---|
| 1000 | 421 | 0 | 11698 | 1657 | 6 |
| 100 | 421 | 0 | 2768 | 1653 | 10 |
| 10 | 421 | 0 | 577 | 1631 | 32 |
| 1 | 421 | 0 | 110 | 1555 | 108 |

**LiNa vs. DUF1646 and nhaA-I**

| E | I | II | III | IV | V |
|---|---|---|---|---|---|
| 1000 | 257 | 164 | 399 | 1618 | 45 |
| 100 | 278 | 143 | 82 | 1576 | 87 |
| 10 | 325 | 96 | 25 | 1388 | 275 |
| 1 | 395 | 26 | 14 | 990 | 673 |

From the table on the left we can see that my combined model overfits to nhaA-I and completely misses hits from DUF1646. This is likely due to an issue with CMfinder; since the nhaA-I search contributes more hits than DUF1646 (1663 > 421) to the combined input to CMfinder, it is likely that CMfinder finds the pattern in nhaA-I and then stops looking for it in DUF1646. This is one problem that motivates the development of the genetic algorithm to replace the CMfinder step. However, we can see that the combined model does find some hits that are not found by either of the individual models, so it still may be useful. Future work will include further analysis of these hits, described in the next section.

From the table on the right we see that the LiNa combined model search also overfits to nhaA-I, but not as badly as the CMfinder combined model. This is likely due to the fact that LiNa uses a different algorithm to combine the models, and it is not as sensitive to the number of hits contributed by each model. The LiNa combined model finds a few hits that are not found by either of the individual models, but not as many as the CMfinder combined model.

### 6. An aside: reporting tandem hits

During model comparison, we also report any instances of tandems, which we define as hits that are located within 100 bp of each other. This information is not useful to our current analysis, but could be an interesting case study and demonstrates the pipeline's breadth of potential functionality. The results are shown below:

```
Found tandem: MGTI01000074.1 11757-11703 11644-11589
Found tandem: MGTI01000074.1 11644-11589 11757-11703
Found tandem: DHQV01000099.1 4515-4574 4379-4429
Found tandem: MVRP01000073.1 1364-1414 1503-1556
Found tandem: MVRP01000073.1 1503-1556 1364-1414
Found tandem: JADJGV010000011.1 59728-59776 59636-59587
Found tandem: JADJGV010000011.1 59636-59587 59728-59776
Found tandem: VGRU01000033.1 489-431 354-296
Found tandem: VGRU01000033.1 354-296 489-431
Found tandem: MVRS01000109.1 9551-9494 9694-9633
Found tandem: DCVI01000207.1 44921-44868 44770-44713
Found tandem: DCVI01000207.1 44770-44713 44921-44868
Found tandem: VGSH01000014.1 30017-29962 30163-30108
Found tandem: VGSH01000014.1 30017-29962 29871-29813
Found tandem: VGSH01000014.1 30163-30108 30017-29962
Found tandem: VGSH01000014.1 29871-29813 30017-29962
Found tandem: MVRP01000001.1 47864-47916 47994-48052
Found tandem: MVRP01000001.1 47994-48052 47864-47916
Found tandem: MVRP01000044.1 25032-24978 25169-25119
Found tandem: MVRP01000044.1 25169-25119 25032-24978
Found tandem: MGTI01000074.1 11757-11703 11641-11590
Found tandem: MGTI01000074.1 11644-11589 11754-11704
Found tandem: MVRP01000073.1 1364-1414 1506-1555
Found tandem: MVRP01000073.1 1503-1556 1367-1414
Found tandem: JADJGV010000011.1 59728-59776 59633-59588
Found tandem: JADJGV010000011.1 59636-59587 59731-59775
Found tandem: VGRU01000033.1 489-431 351-297
Found tandem: VGRU01000033.1 354-296 486-432
Found tandem: QZJR01000105.1 17578-17640 17438-17496
Found tandem: DCVI01000207.1 44770-44713 44918-44868
Found tandem: VGSH01000014.1 30017-29962 30160-30109
Found tandem: VGSH01000014.1 30017-29962 29868-29814
Found tandem: VGSH01000014.1 30163-30108 30014-29963
Found tandem: VGSH01000014.1 29871-29813 30014-29963
Found tandem: MVRP01000001.1 47864-47916 47997-48051
Found tandem: MVRP01000001.1 47994-48052 47867-47915
Found tandem: MVRP01000044.1 25032-24978 25166-25119
Found tandem: MVRP01000044.1 25169-25119 25029-24979
```

## Conclusion

### Discussion

Much time spent working on the project was devoted to learning the basics of bioinformatics and computational biology, as well as the various software packages used in the project. I learned how to use the Infernal, HMMER, CMfinder, and R2R software packages, as well as the command-line interface of the Yale HPC cluster and the Slurm job scheduler to run jobs on the cluster.

Many of our regular meetings were also spent discussing the design of the pipeline, backend database, and code organization. Since I am transitioning the pipeline to SQL, the backend database is currently incomplete, and the code organization is currently in flux. However, I have learned a lot about how to design a modular, extensible, and maintainable software project.

Over the semester of CPSC 290, I've made a great deal of progress on the project, but there is still much to be done before the pipeline is ready for use by the Breaker Lab. I plan to continue working on the project over the summer, by the end of which I hope to have a working prototype. I will continue my experiments on sodium-binding riboswitches with the pipeline over the summer as well, and I hope to have some interesting results to share when I'm done.

### Future Research

I hope to create a platform to facilitate and democratize biological sequence-based research by automating key processes that currently require installing and running command-line tools and working with filesystems manually.

There are some steps to the pipeline that have not yet been implemented, but are planned for the future. These include:

- [ ] Completing the transition to SQL and implementing a new `cmsearch_compare` method to compare two searches using SQL queries to identify duplicate/overlapping hits and tandem repeats in the outputs of `infernal` searches.
- [ ] A function to map the hits of a `cmsearch` search onto the phylogenetic tree contained in the Genome Taxonomy Database (GTDB) using R or Python to read the GTDB taxonomy tree and the Infernal output files, and then render a new file with the hits mapped onto the tree. This will enable us to view the distribution of hits across the tree and determine if there are any patterns in the distribution of hits that may be useful for identifying new riboswitches. For example, if a particular riboswitch is found in a particular clade of bacteria, it may be possible to use this information to identify new instances of the riboswitch in other, closely related bacteria. Or, if two riboswitches, such as nhaA-I and DUF1646, are found in the same clade of bacteria, this may indicate that the two riboswitches have common ancestry and can be combined into a single model. This is one method of determining whether a "false positive" hit from the combined search is actually a true positive that was missed by the original models, and therefore should be added to the model.

- [ ] A function to access genetic context for a sequence of interest from the GTDB FASTA file and render a visualization of the genetic context of the sequence of interest. This will enable us to view gene associations of a given sequence, such as a cmsearch hit, and determine if the sequence is associated with a particular gene or gene cluster, which can tell us valuable information. For example, if a hit is upstream of a gene that is annotated as a transporter (e.g. a sodium ion transporter), it is likely that the sequence is a riboswitch that regulates the expression of that transporter gene. This is another method of determining whether a "false positive" hit from the combined search is actually a true positive that was missed by the original models, and therefore should be added to the model.

In addition to these planned features, I plan to use the pipeline to update the Breaker Lab's collection of riboswitches. The Breaker Lab has a roster of ~50 riboswitches that have been identified in bacteria and eukaryotes, but have not been updated for a number of years. I am planning to utilize the pipeline to update the nhaA-I and DUF1646 motifs and other riboswitches using updated data, in order to identify more instances of these riboswitches and improve the models.

## Acknowledgements

I would like to thank my mentor, Chris, for his guidance, support, and instruction throughout the semester. I would also like to thank Dr. Ronald Breaker and the Breaker Lab for giving me the opportunity to work on this project.