A framework for comparing genome annotation from several sources

Aila Aspås
Master’s Thesis Autumn 2017
A framework for comparing genome annotation from several sources

Aila Aspås

Wednesday 1st November, 2017
Abstract

Annotation of genomes is an important process which increases the usefulness of genomic data immensely. However, there are many annotation tools in use, and they can all give different results. This master project is about creating a framework for comparing annotations from various sources. This is done in the context of annotation of ribosomal RNAs, or rRNA, in genomes. Knowledge of the rRNA and its location in a genome is very useful for working out evolutionary relationships among organisms. This because they are of ancient origin and are found in all known forms of life. So there is a lot to gain by finding them. The program being made gives the possibility to compare and analyze the available as well as future predictor programs. This is useful for making a choice of which to make use of and a helpful tool for continued development of these programs.
# Contents

1 Introduction  
2 Background  
  2.1 Genetics  
  2.1.1 Genome  
  2.1.2 DNA  
  2.1.3 Protein  
  2.1.4 RNA  
  2.2 Ribosome  
  2.3 Phylogeny  
  2.4 Phylum  
  2.5 Genome annotation  
  2.6 Hidden Markov Models  
  2.7 Structure and sequence alignment  
  2.8 Notable file formats  
  2.8.1 Gbk file  
  2.8.2 Fasta file  
  2.8.3 GFF file  
3 Applications  
  3.1 Profile hidden Markov model and HMMER  
  3.2 RNAmmer  
  3.3 PostgreSQL  
  3.4 BioSQL  
  3.5 Python  
  3.5.1 Biopython  
  3.5.2 Psycopg  
  3.5.3 Matplotlib and Seaborn  
4 The project  
  4.1 Goal  
  4.2 Objectives  
5 Methods  
  5.1 The data  
  5.2 NCBI and structure of genome data  
  5.2.1 Sequence Identifiers  
  5.2.2 GI numbers  

v
# List of Figures

2.1 Visualization of DNA vs RNA ........................................... 3
2.2 Biological classification. Credit: Peter Halasz ...................... 6
2.3 Visualization of global and local alignment ...................... 8
2.4 Picture from Center for Molecular biology of RNA: Showing Ecoli 23s secondary structure. ................................. 8

3.1 Profile-HMM from HMMer (the software) 2.3.2 user guide . 12

5.1 The parts of the bioSQL schema most relevant, in picture biodatabase = sub-biodatabase ....................................... 24
5.2 Flowchart main program ................................................. 29

6.1 Alphaproteobacteria plot data (n= 14) ............................... 32
6.2 Betaproteobacteria plot data (n= 9) ................................. 34
6.3 Gammaproteobacteria plot data (n= 34) ............................ 34
6.4 Actinobacteria plot data (n= 14) .................................. 35
6.5 Bacteroidetes plot data (n= 4) ...................................... 35
6.6 Chlamydiae plot data (n= 3) ........................................ 36
6.7 Cyanobacteria plot data (n= 3) ...................................... 36
6.8 Firmicutes plot data (n= 27) ......................................... 37
6.9 Spirochaetes plot data (n= 3) ....................................... 37
6.10 Tenericutes plot data (n= 3) ....................................... 38
6.11 All plot data (n= 126) ............................................... 38

10.1 BioSQL schema overview .............................................. 48
List of Tables

2.1 Table over ribosomal RNA units . . . . . . . . . . . . . . . . 5
6.1 Run times . . . . . . . . . . . . . . . . . . . . . . . . . . . . 31
6.2 Table over taxonomic category: phylum . . . . . . . . . . . 33
Acknowledgements

I would first like to thank my supervisor Karin Lagesen for all of her help and guidance, without you I really could not have done this. All of my friends and family for their support and help, I am very luck to have all of you. A special thanks to my mom for some proofreading and my aunt Ulrika for providing excellent studying environment.

Aila Aspås
University of Oslo
November, 2017
Chapter 1

Introduction

Bioinformatics is a combined academic field that develops methods and software tools for understanding biological data. This includes processing and annotating genomes and its subparts such as the Ribosome, DNA, protein and so on.

The ribosome has the job of biological protein synthesis (protein translation). Ribosomal ribonucleic acid (rRNA) is a part of the ribosome. The ribosome, including rRNA, is in all living organisms which makes rRNA ideal for work aimed to explain the evolutionary relationships between species. Ribosomal RNA is also distinct enough to separate species without being to variable (susceptible to mutation) to recognize and categorize. Distinct as in rRNA has a structure that is evolutionarily conserved, of ancient origin.

For rRNA to be useful in evolutionary studies it needs to be accurately annotated, because small changes in an analyzed sequence of an rRNA can have drastic consequences on its interpretation. Since DNA sequences have a limited library (only four letters) random matches that are misleading or insignificant can occur when reading the sequence. To avoid some of these it is possible to preform both structural alignment and sequence comparison together. Programs such as BLAST that uses only sequence comparison alignment are therefor not always sufficiently accurate at predicting locations of rRNAs. In the past rRNA prediction based only on sequential alignment went so wrong that they concluded some genomes did not have any rRNAs annotated at all and other genomes seem to have rRNAs annotated on the wrong strand. Both well known faults.

Lagesen, 2008

So there was, and still is, a need for programs aimed at accurately annotate rRNAs. RNAmmer is such a program made to help do these annotations. The program task is to predict the units of rRNA in full genome sequences and returning readable results. HMMer is a module uses in RNAmmer for creating Hidden Markov Models that helps finding rRNAs. It makes use of both sequential and structural alignment.
RNAmmer has shown to be a popular predictor program and it will be the one used to built this analytic framework. It is worth to note that other prediction programs can have replacement for the role HMMer has in RNAmmer with other modules. These module would most likely still make use of Hidden Markov Models, as HMMer does, although other possibilities needs to be considered. The framework should be able to give insight into the accuracy between several predictor programs and also be of help in judging accuracy pertained in updated versions.

To be able to build this framework and get untainted results, new training data is needed to run the computational finder module and a output interpreter add to make more clear the results. In other word raw data, that has not been obtained through the running of predictors.
Chapter 2

Background

The theoretical bases needed to get a good grasp of the framework and its development.

2.1 Genetics

Ribonucleic acid (RNA) is a polymeric molecule. It is involved in various biological roles in coding, decoding, regulation, and expression of genes. DNA and RNA are nucleic acids, and, along with proteins and carbohydrates, constitute the three major macromolecules essential for all known forms of life (Lagesen, 2008). As mentioned, Ribonucleic acid (RNA) is one of a lifeform’s main building blocks along with DNA, proteins and carbohydrates. The role RNA, and there by in part rRNA, has is closely connected to both DNA and proteins. So to understand RNA a need arises to understand some basic of the others building blocks of life as well.

Figure 2.1: Visualization of DNA vs RNA
2.1.1 Genome

A genome is the genetic material of an organism. It consists of DNA. The genome includes both the genes (the coding regions), the noncoding DNA and the genetic material of the mitochondria and chloroplasts.

2.1.2 DNA

Deoxyribonucleic acid (DNA) is a molecule that stores the genetic instruction needed for all living organisms to develop and function. It consists of two strand coiled around each other to form a double helix, each strand is a chain of small molecules called nucleotides. Each nucleotide is composed of a nitrogen-containing nucleon base - either guanine (G), adenine (A), thymine (T), or cytosine (C) - as well as a monosaccharide sugar called deoxyribose and a phosphate group. The strand are bonded together with the paring of A with T and C with G. These four letters are also how DNA sequence is represented digitally. Usually, DNA is tightly wound and forms a chromosome. The total information stored in all chromosomes constitute a genome.

2.1.3 Protein

Proteins perform a vast array of functions within living organisms, including catalyzing metabolic reactions, replicating DNA, responding to stimuli, and transporting molecules from one location to another. They are built up by amino acids. Depending on their different chemical properties to decide the proteins shape and function. Proteins can form structural elements and also act as a catalyst in the cell.

2.1.4 RNA

RNA has much the same strand construction as DNA, but with uracil (U) instead of thymine (T), and it is made up of a single-strand folded unto itself. The secondary structures of biological DNA and RNA tend to be different because of the double helix for DNA versus the single strand for RNA. Biological DNA mostly exists as fully base paired double helices, while biological RNAs single strands and often forms complicated base-pairing connections due to increased ability to form hydrogen bonds. Nucleic acid secondary structure is the base pairing interactions within a single nucleic acid polymer or between two polymers. It can be represented as a
list of bases which are paired in a nucleic acid molecule. Some RNA molecules play an active role within cells by catalyzing biological reactions, controlling gene expression, or sensing and communicating responses to cellular signals. One such role is played out in the ribosome.

### 2.2 Ribosome

The ribosome is a large and complex molecular machine, found within all living cells, that serves as the site of biological protein synthesis (translation). Ribosomal ribonucleic acid (rRNA) is the RNA component of the ribosome, and is essential for protein synthesis in all living organisms. Protein synthesis is a function whereby mRNA molecules direct the assembly of proteins on ribosomes. This process uses transfer RNA (tRNA) molecules to deliver amino acids to the ribosome, where ribosomal RNA (rRNA) links amino acids together to form proteins. The ribosomes function is dependent on its rRNA, so information on both the sequence and the structure level of the rRNA is important to know. The ribosome consists of two subunits, the small and the large subunit, which pair up with several ribosomal proteins to form the functional ribosome. The rRNAs present in prokaryotes are the 5S and 23S in the large subunit, and the 16S in the small subunit. In eukaryotes, 5S, 5.8S and 28S rRNA exist in the large subunit, and 18S rRNA in the small subunit. Due to being present in all living things, rRNA sequences have been used a lot to explain the evolutionary relationships between species.

### 2.3 Phylogeny

Phylogeny is a way of representing the history of the evolution of an organism in reference to lines of descent and relationships among a large groups of organisms. Phylogenetics is the analysis of macromolecular sequence data, originally of morphological data, that is used to deduce such evolutionary relationship between and for all organisms on earth. From, phylogeny we get information that can be used to develop phylogenetic maps of relation, phylogenetic trees. Originally these tree were create based on the physical attributes we could physically see on an organism, which was not ideal. The evolutionary branching process may be illustrated in these trees, out from the placement of organisms and its branches.
Later a paper by Woese and Fox (Woese and Fox, 1977) shed light on phylogenetic relationships based on quantitative analysis on gene sequences. The paper also change the view on grouping of organisms. It established that all cellular life falls into one of three (not two as believed earlier) large relatedness groups. Eukaryotes (human types of cells, which contain a nuclear envelope), eubacteria (this group is where classically studied bacteria fit), and the new archaebacteria (an unusual group organisms named to distinguish the group from eubacteria) (Pace, Sapp and Goldenfeld, 2012).

2.4 Phylum

In biology, a phylum (or multiple phyla) is a level of classification or taxonomic (see figure nr 2.2 place four from top). Phyla can be thought of as groupings of organisms based on general specialization of body plan. At its most basic, a phylum can be defined in two ways. One, as a group of organisms with a certain degree of morphological or developmental similarity (the phenetic definition), or a group of organisms with a certain degree of evolutionary relatedness (the phylogenetic definition).

2.5 Genome annotation

The task of primary analysis of genomes, or genome annotation, is the process of finding and designating locations of individual genes and other features on raw data. Once a genome is sequenced, it needs to be annotated to make sense of it. The simplest way to perform gene annotation uses homology based search tools, like BLAST, to search for homologous genes in specific databases, the resulting information is then used to annotate genes and genomes. RNAmmer is also such a search tool that will be described in more detail later in this text. It possible to form a clearer picture of genome annotation by diving it up a little. Thinking of it as either structural or functional annotation. Structural consists of the identification of genetic elements. Functional annotation consists of attaching biological information to genetic elements.
2.6 Hidden Markov Models

A hidden Markov model (HMM) is a statistical model that describes the transition between visible tokens that depend on an assumption of the existence of hidden transition states as well. An HMM consists of two stochastic processes, one for the visible tokens (output) and another of the hidden states. The hidden states form a Markov chain, and the probability distribution of the observed tokens depends on the underlying state. Modeling observations in these two layers, one visible and the other invisible, is very useful, since many real-world problems deal with classifying raw observations into a number of categories, or class labels, that are more meaningful to us. For example, a biological sequence consists of smaller substructures with different functions, and different functional regions often display distinct statistical properties. For example, it is well-known that proteins generally consist of multiple domains. Given a new protein, it would be interesting to predict the constituting domains (corresponding to one or more states in an HMM) and their locations in the amino acid sequence (observations).

2.7 Structure and sequence alignment

A sequence alignment is usually the best way of comparing sequences. Through such an alignment one can get an indication whether or not two sequences are homologous (have a common evolutionary ancestor), details on the genomes function can be derived, and biological hypotheses may be created.

One can use either global or local sequence alignment. Global alignment is entire sequences compared against each other and is used when the sequences are of approximately equal length and are expected to be related. Local alignment is of subsequences (part of a whole) from each sequence and is used when the goal is to identify which parts of the sequences should be included, when the sequences are of unequal length and/or only certain regions in the sequences are assumed to be related (conserved domains).

Other options are a combination of these two, hybrid methods, known as semi-global or "glocal" (short for global-local) methods, attempt to find the best possible alignment that includes the start and end of one or the other sequence. A case where semi-global alignment is useful is when one sequence is short and the other is very long. Then the short sequence should be globally aligned but only a local alignment is desired for the long sequence.

There are some limitations in sequence alignment and its accuracy. For example, when analyzing RNA and DNA, one problem that can occur is that since both RNA and DNA are represented in sequences with a limited library of four letters they will have a higher rate of random matches of no
significance, compared to protein sequences where there are 20 letters and less chance for random repetition and "lucky" meaningless matches. In such cases one can also make use of the secondary (mentioned under RNA) or tertiary structure to identify homologous parts (evolutionary relationships). In other word structure alignment. This method can use two or more polymer structures based on their shape and three-dimensional conformation sequences. This method can only be used on sequences where these structures are known, because these alignments rely on information about all the query sequences secondary structure conformations. That results in that structural alignments can be more reliable between sequences that are very distantly related and that have diverged so extensively that sequence comparison cannot reliably detect their similarity.

2.8 Notable file formats

In the making and running of this program the data was stored on files of different types. The gbk file format is how the test data were obtained in and stored before use. Fasta is the plain text format the sequence need to be in for the predictor program (RNAmmer) to read and run it. GFF
is the output format from the predictor program. All of the format comes in to play in this project. It is also worth noting that the database schema BioSQL can take in data via biopython in many other format such as FASTA because there are raedy made parsers. But there was a need to make a gff file reader since there was not a parser ready of that format.

2.8.1 Gbk file

The GenBank (gbk) format was developed by the U.S. National Center for Biotechnology Information (NCBI). It is a file format used for storing the information pertaining to a genome and save its sequences. The whole file is in plain text format. It designed to be easily parsed by computer programs, this because of the fields containing different types of information are well-labeled. The header of the file contains information describing the sequence, such as its type, shape, length, and source.

```plaintext
LOCUS   NC_014644   1667350 bp    DNA    circular  CON 03–AUG–2016
DEFINITION Gardnerella vaginalis ATCC 14019 chromosome, complete genome.
ACCESSION NC_014644
VERSION   NC_014644.1
DBLINK   BioProject: PRJNA55487
          Assembly: GCF_000159155.2
KEYWORDS RefSeq.
SOURCE   Gardnerella vaginalis ATCC 14019
          Bacteriaceae; Actinobacteria; Bifidobacteriales; Bifidobacteriaceae; Gardnerella.
          ...
          ...
          ...
CDS complement(234251..235033)
  /locus_tag="HMPREF0421_20205"
  /EC_number="3.5.1.–"
  /note="COG: COG0846; Pfam: PF02146; InterPro: IPR003000"
  /codon_start=1
  /transl_table=11
  /product="NAD-dependent deacetylase"
  /protein_id="YP_003985314.1"
  /db_xref="GeneID:9903745"

gene 235611..237126
  /locus_tag="HMPREF0421_r20001"
  /db_xref="GeneID:9903746"

rRNA 235611..237126
  /locus_tag="HMPREF0421_r20001"
  /product="16S ribosomal RNA"
  /db_xref="GeneID:9903746"
```

(Above example of the first couple of line in a GenBank file and an excerpt of several lines down showing features in the same file)

2.8.2 Fasta file

In bioinformatics, the FASTA file format is a text-based format for representing either nucleotide sequences or peptide sequences. The nucleotides and amino acids are represented using single-letter sequences. The format also allows for sequence names and comments to precede the sequences. FASTA is pronounced "fast A", and stands for "FAST-All",...
because it works with any alphabet, an extension of "FAST-P" (protein) and "FAST-N" (nucleotide) alignment

2.8.3 GFF file

The general feature format (gene-finding format, generic feature format, GFF) is a file format used for describing genes and other features of DNA, RNA and protein sequences. The filename extension associated with such files is .GFF and the content type associated with them is text.

```
## gff-version 2
## source-version RNAmmer-1.2
## date 2017-09-28
## Type DNA
# seqname source feature start end score +/- frame attribute
#
NC_012659.1 RNAmmer-1.2 rRNA 268387 271306 3600.0 + . 23s_rRNA
NC_012659.1 RNAmmer-1.2 rRNA 4652151 4655070 3598.5 - . 23s_rRNA
NC_012659.1 RNAmmer-1.2 rRNA 271358 271472 97.6 + . 5s_rRNA
NC_012659.1 RNAmmer-1.2 rRNA 4651933 4652047 94.6 - . 5s_rRNA
NC_012659.1 RNAmmer-1.2 rRNA 266694 268233 2076.6 + . 16s_rRNA
NC_012659.1 RNAmmer-1.2 rRNA 4655224 4656763 2072.5 - . 16s_rRNA
#
```

(Above a part of a gff produced by running RNAmmer-1.2 in the project)
Chapter 3

Applications

The programs (tools) that are used to make this framework possible to develop, run and otherwise influence this project.

3.1 Profile hidden Markov model and HMMER

Profile hidden Markov is a part of the predictor RNAmmer that has such a significant part in the resulting output that not taking its task handling in to account might give a wrongful picture on the result. In other words, the result from RNAmmer are affected by the Profile hidden Markov.

Profile hidden Markov turns a multiple sequence alignment into a position-specific scorings system by the use of profiles. The profile hold certain information on positions in a family about areas that are more conserved than other, areas that are less allowing for substitutions. This is done by that HMMER holds the area information by taking a look at pre-acquired subsequences, that holds known structures and conserved areas (a family), and creating a method for comparing and determining if a new subsequence is in a give family. By keeping positional data in profiles and combining with the hidden Markov models ability to position specify gap penalties we get to capture more of the important information about the degree of conservation at various positions, and the varying degree to which gaps and insertions are permitted.

HMMER is an implementation tool, for UNIX platform, of profile hidden Markov model methods for sensitive searches of biological sequence databases using multiple sequence alignments as queries.

Figure 3.1 is the Plan7 profile-HMM employed in the HMMER package for scans of nucleotide or polypeptide sequences. Most transitions are assigned scores (not shown).

The advantage of using HMMs is that HMMs have a formal probabilistic basis. Probability theory is used to guide how all the scoring parameters should be set. This probabilistic basis makes it possible work with things
that more heuristic methods cannot do easily. For example, a profile HMM can be trained from unaligned sequences. Another consequence is that HMMs have a consistent theory behind gap and insertion scores. In most details, profile HMMs are a slight improvement over a carefully constructed profile and less skill and manual intervention are necessary to use profile HMMs. This allows for creating libraries of hundreds of profile HMMs and that can be applied on a very large scale to whole genome analysis. In other words. Given a multiple sequence alignment as input, HMMer builds a statistical model called a "hidden Markov model" which can then be used as a query into a sequence database to find (and/or align) additional homologues of the sequence family.

### 3.2 RNAmmer

RNAmmer is a computational predictor (an genome annotation tool) for the major rRNA types from all kingdoms of life. An aim with this program is to make it easier to create accurate annotations for ribosomal RNA (rRNA).

What inspired the making of RNAmmer was a clear lack of sufficient annotation of rRNA in genome databases. Annotation is the process by which pertinent information about raw sequences is added to the genome databases. This involves describing different regions of the code and identifying it. It was found that genomes sometimes did not have any rRNAs annotated at all, other genomes seem to have rRNAs annotated on the wrong strand.

RNAmmer is a program that has an internal module that is first trained on pre-acquired datasets, then run in two phases. The first phase is the spotter, where areas of the sequence is search to find where rRNA is most likely to be. The second phase is a more exact search in these areas to get
the rRNAs and precise data on them.

The first module in RNAmmer is for now trained on data from the 5S ribosomal RNA database and the European ribosomal RNA database project. The alignments are all structural alignments i.e. aligned using secondary structure information gained from comparative sequence analysis. The resulting data was divided into separate alignments for archaeal, bacterial and eukaryotic sequences. These groups provides a kingdom-specific predictors. This pre-screening step makes the method fast with little loss of sensitivity.

RNAmmer uses Hidden Markov Models to get speed and accuracy. The HMM is a part of the incorporated HMMer module. HMMs represents match states as columns with a specific nucleotide distribution, deletion states represent the possibility of gaps, and insertion states represent columns with large numbers of gaps. Transition probabilities between the states indicate how likely each of the states are. With this it's possible to work with the likelihood of insertions and deletions varying along a sequence depending on if a part is conserved or variable. A good match to the HMM may come either from a highly conserved region which may be short, or from a longer region with only weak conservation.

Columns at the beginning and end of the multiple alignments often have low conservation and many gaps. Such columns are often adjusted into the HMM as insert states, but HMMer ignores them at the beginning and end of the alignment. This may lead to misplacement of match sequences start and stop state which can lead to inaccuracy in endpoint prediction. So RNAmmer is reasonably good at finding rRNA, but has start and stop prediction deviations.

### 3.3 PostgreSQL

Why a database? A database comes in use when an application is going to handle large amount of data, either under the development stage or as part of its end function. This because when working with large quantities of data it needs to be store, often multiple times, as well as a need for the storing to be such that the data can an easily be accessed. In other words a good databases is essential component when developing a program (or application) intended for handling large amounts of data.

A general description of any database is that it is a large organized collection of data organized so that the information it contains can be easily accessed, manage and updated. Then there is relational databases, which stores data by using a collection of schema, tables, queries, reports, views, and other elements. This is to make it even easier to find and retrieve relevant information, because it makes it possible to traverse the data with
just a few key words instead of reading through it all in search of relevant information. Data once in the database is not "locked" and can still get updated, expanded, and deleted so new and more relevant information can be added. Databases process workloads to create and update themselves, querying the data they contain and running applications against it. Now the data can come in various format, shape, and size, and the way it is handled varies largely based on its content and intended use. The intended use is also relevant when choosing of database for storing.

PostgreSQL is said to be a relational database management system (RDBMS), which can be seen as an extension of the "plain" Database management system (DBMS). The key difference to note is that RDBMS (relational database management system) applications store data in a table form, while DBMS applications store data as files. DBMS and RDBMS are sets of programs that have the purpose to handle the details pertaining to storing and managing data. They are able to interact with end-users, other applications, and the database itself to capture and analyze data. The above mentioned differences are less relevant when the two types are "merge" which is becoming more common, and that is actually more of the case of PostgreSQL now.

Another plus points for PostgreSQL is that it is open source software, free, and by all accounts is "locked" at reliable and stable in its development parallel to other DBMS, both commercial and open sourced. PostgreSQL is highly customizable (can be fitted after necessity), it runs stored procedures in more than a dozen programming languages, including Java, Perl and Python. Also library interfaces allowing the various languages (above mentioned) compiled and interpreted to interface with PostgreSQL, notable python and biopython. Included are also a framework that allows developers to define and create their own custom data types along with supporting functions and operators that define their behavior.

3.4 BioSQL

BioSQL is a joint effort between several open bio-focused programming projects, aiming to build and support a shared database schema for storing sequence data. The module is a generic relational model covering sequences, features, sequence and feature annotation, a reference taxonomy, and ontologies. BioSQL has expanded from its start as a local relational storage for GenBank, to becoming a collaboration between the BioPerl, BioPython, BioJava, and BioRuby projects, with a goal to build a sufficiently generic schema for persistent storage of sequences, features, and annotation in a way that is interoperable between the Bio projects. Supported RDBMs are at present PostgreSQL, MySQL, Oracle, HSQLDB, and Apache Derby for the core schema. The collaboration means that it is possible to load a GenBank file into the database with BioPerl, then use Biopython to extract the file from the database as a record object including its
features, then get out more or less the same information as if the GenBank file had been directly loaded in as a SeqRecord using SeqIO.

3.5 Python

Python is a high-level, object-oriented programming language for general-purpose programming. Created by Guido van Rossum. Python has a design philosophy that emphasizes code readability. It has simple easy-to-use syntax than allows for fewer lines than might be used in other languages such as C++ or Java, notably it use whitespace indentation rather than curly brackets. The language provides constructs intended to enable writing clear programs on both a small and large scale.

3.5.1 Biopython

Biopython is a set of libraries and tools for biological computation written in Python. The Biopython project that is its basis is an open-sourced project by an international team of developers, and is also a member project of the Open Bioinformatics Foundation (OBF).

The goal of Biopython is to make it as easy as possible to use Python in connection with bioinformatics programming, this by creating high-quality, reusable modules and classes. Biopython features include (with focus on relevance to this project) parsers for Bioinformatics file formats (including FASTA and Genbank files), access to online services (NCBI..) and integration with BioSQL, the sequence database schema.

The Biopython project has also made many parsers aimed at transferring the various format biological data comes in, like fasta and gbk files. A parser for gff files is under development, but as of the end of this thesis no such parser is included in the biopython package. Note that the gff parser code being developed is available for reading while its being built, for inspiration.

3.5.2 Psycopg

Psycopg is the most popular PostgreSQL adapter for the Python programming language. Psycopg 2 is mostly implemented in C as a libpq wrapper, making it both efficient and secure. It features client-side and server-side cursors.

3.5.3 Matplotlib and Seaborn

Matplotlib is a Python 2D plotting library which aims to produces publication quality figures. Matplotlib can be used in Python scripts, the Python and many others. It make it possible to generate plots, histograms, power spectra, bar charts, error charts, scatter plots, etc., with just a few
Seaborn is a Python visualization library based on matplotlib. It provides a high-level interface for drawing statistical graphics. It is built on top of matplotlib and tightly integrated with the PyData stack, including support for numpy and pandas data structures. Seaborn aims to make visualization a central part of exploring and understanding data. The plotting functions operate on data frames and arrays, containing a whole dataset and internally performs the necessary statistical model-fitting to produce informative plots. Seaborn should be thought of as a complement to matplotlib, not a replacement for it.

If matplotlib “tries to make easy things easy and hard things possible”, Seaborn tries to make a well-defined set of hard things easy.
Chapter 4

The project

4.1 Goal

In this project the aim is to make a program that takes a rRna predictors (example RNAmmer), runs it and document its results compared to raw data. Or simply a framework for comparing genome annotation from several sources. Specifically building a program that take in raw data in Genbank files format (gbk), parses them, and input the data into a database (postgresql with bioSQL), then uses that data's sequences to run new prediction on rRna location with chosen predictor. The new results are read into the database. Then finally the program extracts the raw data and the new data and compares its accuracy and presents it in plots. This all with the aim to look at accuracy based on the choice made in the different built of predictor, for example in the use of structure alignment used in the comparison process.

4.2 Objectives

List of separate tasks that need to be achieved to get the whole to work.

- Read and load in a GenBank file into the database.
- Get the predictor to run and give an output.
- Read and load in a gff file into the database
- Retrieve specific data from the database
  - bioentry based on accession number
  - biontries features
  - a features location and attribute
• Make a program that can sort and compare the data
• Make some kind of visualization of compared data
Chapter 5

Methods

5.1 The data

There are 119 files in the catalog (named refseq) with all the test genomes. In all 239 genomes in all, but since several are plasmid and of no interest they are not read in to the the BioSQL schema by loadDB. The number of genome read in is 129. In the first round all was read in into one sub-biodatabase named after the main folder "refseq" then run through the rRNA estimator (RNAmmer) and returned in a new sub-biodatabase named after the estimator(RNAmmer-1.2). The amount of genome that are in RNAmmer-1.2 is 126, the 3 "lost" are and the 7 more then original amount are most likely genomes with two chromosomes where the rRNA has ended up some times on both (increasing the amount of results) or on only one (resulting in an input without rRNA).

Under the biodatabase "refseq" and "RNAmmer-1.2" are all the test data before and after running the prediction program the first time. The other are the sub- biodatabases based on the phylum that are again divided into before and after prediction. In total there are 17 phylum categories but the proteobacteria have been split in to 5 sub groups (alpha, beta...), so the final count is 22 times 2 for before and after plus the to complete biodatabases. So there are 46 in all.

<table>
<thead>
<tr>
<th>biodatabase_id</th>
<th>name</th>
<th>authority</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>106</td>
<td>Actinobacteria</td>
<td></td>
<td></td>
</tr>
<tr>
<td>103</td>
<td>Alphaproteobacteria</td>
<td></td>
<td></td>
</tr>
<tr>
<td>114</td>
<td>Aquificae</td>
<td></td>
<td></td>
</tr>
<tr>
<td>116</td>
<td>Bacteroidetes</td>
<td></td>
<td></td>
</tr>
<tr>
<td>102</td>
<td>Betaproteobacteria</td>
<td></td>
<td></td>
</tr>
<tr>
<td>115</td>
<td>Chlamydiae</td>
<td></td>
<td></td>
</tr>
<tr>
<td>104</td>
<td>Chlorobi</td>
<td></td>
<td></td>
</tr>
<tr>
<td>117</td>
<td>Chloroflexi</td>
<td></td>
<td></td>
</tr>
<tr>
<td>107</td>
<td>Cyanobacteria</td>
<td></td>
<td></td>
</tr>
<tr>
<td>113</td>
<td>Deinococcus–Thermus</td>
<td></td>
<td></td>
</tr>
<tr>
<td>108</td>
<td>Deltaproteobacteria</td>
<td></td>
<td></td>
</tr>
<tr>
<td>119</td>
<td>Dictyoglomi</td>
<td></td>
<td></td>
</tr>
<tr>
<td>111</td>
<td>Epsilonproteobacteria</td>
<td></td>
<td></td>
</tr>
<tr>
<td>101</td>
<td>Firmicutes</td>
<td></td>
<td></td>
</tr>
<tr>
<td>105</td>
<td>Fusobacteria</td>
<td></td>
<td></td>
</tr>
<tr>
<td>100</td>
<td>Gammaproteobacteria</td>
<td></td>
<td></td>
</tr>
<tr>
<td>118</td>
<td>Nitrospirae</td>
<td></td>
<td></td>
</tr>
<tr>
<td>121</td>
<td>Planctomycetes</td>
<td></td>
<td></td>
</tr>
<tr>
<td>94</td>
<td>RNAmmer-1.2</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
5.2 NCBI and structure of genome data

5.2.1 Sequence Identifiers

Many sequences have two types of identification numbers, GI and VERSION (ACCESSION number plus version number). The two identifier types are different in format, and were implemented at different times but are both unique for each version of a sequence data.

5.2.2 GI numbers

A GI number (for GenInfo Identifier, sometimes gi) is simple series of digits that are assigned to each sequence record in connection with NCBI. The GI number is in no way similar to the version number of the sequence record. Each time a sequence record is changed, the sequence record is assigned a new GI number. A nucleotide sequence GI number is shown in the VERSION field of the database record. A protein sequence GI number is shown in the VERSION field of a gbk record.

5.2.3 Sequence Versions

A sequence Version gathers all of the GI numbers for a specific sequence into ordered series. A sequence version number consists of a base Accession number, a dot, and a version nr. (This identifier is often referred to as an “accession dot version.”) The base Accession number identifies the sequence record, and the version nr gives which of the version. A sequence
Accession number without a version nr is the newest version of the sequence.

The two systems of identifiers run in parallel to each other. In other words, when any change is made to a sequence, there will be a new GI number, and the version part of its accession number is increased by 1.

5.3 The database

Distinction that is need here: commands in postgreSQL happen in bioSQL Schema but it is commands written in postgreSQL. So when demonstrating command used it is in postgreSQL, but if it is to demonstrate the bioSQL (the schema) structure or in connection with its role the command will be shown under bioSQL.

5.3.1 PostgreSQL use

PostgreSQL is a client/server model. A server managing the database, accepts connection from a client application, and performs database actions on behalf of a clients. This can be applied on one and the same computer, or on one computer serving as server and one (or more) computer as client. For a setup with two (or more) computers some connecting program needs to be installed on both client and server. The programs needed in this projects setup is the communication language biopython helped by psycopg2 the PostgreSQL adapter.

UiO logged on to its server (database hotel) Command line to log in and accessing the database (rnammer):

```
psql -h dbpg-ifi-utv.uio.no -p 5432 -U ailaka -d rnammer
```

then add password. Command used for going directly in to the database and orienting.

```
=> \ h % syntax help
=> \ q % quit SQL
=> \ d % list of all relations (tables)
```

For looking up all biodatabases and see if they have been created. The symbol "*" equals "all".

```
SELECT * FROM table_name;
% to see all entries for each col
```

Example from early in development. Command and output in terminal.

```
rnammer=> select * from biodatabase;
```
For looking closer at two of the bioentries. Command and output in terminal.

```
rnammer=> SELECT DISTINCT bioentry.accession
    FROM bioentry JOIN biodatabase USING (biodatabase_id)
    WHERE biodatabase.name = 'rnammerTEST';
```

```
<table>
<thead>
<tr>
<th>accession</th>
</tr>
</thead>
<tbody>
<tr>
<td>gi</td>
</tr>
</tbody>
</table>
```

(1 row)

```
rnammer=> SELECT DISTINCT bioentry.accession
    FROM bioentry JOIN biodatabase USING (biodatabase_id)
    WHERE biodatabase.name = 'genebank';
```

```
<table>
<thead>
<tr>
<th>accession</th>
</tr>
</thead>
<tbody>
<tr>
<td>NC_000913</td>
</tr>
<tr>
<td>NC_012659</td>
</tr>
<tr>
<td>NC_012796</td>
</tr>
<tr>
<td>NC_013062</td>
</tr>
<tr>
<td>NC_015846</td>
</tr>
<tr>
<td>NC_017146</td>
</tr>
</tbody>
</table>
```

(6 rows)

Sorting alternatives worth noting alphabetically.

```
SELECT * from biodatabase ORDER BY biodatabase_id ASC;
```

Numeric sorting of entry number

```
SELECT * from biodatabase ORDER BY biodatabase_id DESC;
```

### 5.3.2 BioSQL use

**Biodatabase**

A biodatabase is simply a collection of bioentries (read own section). One bioentry can only be in one biodatabase, but one biodatabase can contain many bioentries. Biodatabase entities can be identified by their name, example "RNAmmer-1.2", "Actinobacteria". Databases may also be further identified by an authority, but that was not used in this case.
Bioentry

A bioentry is any single entry or record in a biological database, it is the core entity of the BioSQL schema. The bioentry contains information about the record name, accession nr and version, its description and an identifier nr. For working convenience with GenBank records, the division of GenBank can be specified in a 3 character field. Example of data as a genbank file structure under and then how it looks when entered into the bioSQL schema as a seqRecord.

```
LOCUS NC_000913 4641652 bp DNA
circular CON 15–MAY–2014
DEFINITION Escherichia coli str. K–12 substr. MG1655, complete genome.
ACCESSION NC_000913
VERSION NC_000913.3 GI:556503834
... (Above is a part of Escherichia coli gbk file)
name: NC_000913.3
accession: NC_000913
identifier: 556503834
division: CON
description: NDP=Norrie disease {first three exons, microdeletion regions}
version: 3
... (Above is a part of Escherichia coli seqRecord read in terminal)
```

It is worth noting that bioentries need not come from a public database. This transfer from gbk file to bioentry is what happen in the first step of the code run to compile the project, loadDB.

Structure and content of BioSQL schema

The SeqRecord object used in BioPython holds a sequence (as a Seq object) with identifiers (ID and name), description and optionally annotation and sub-features. Note that getting SeqRecord is dependent on the accession number (first part of the version number), even if version number and GI number is also in the record. This can potentially make some problems since a genome can have several versions.

Although the upload data have a wide variety of information connected to one genome sequence, the main information needed in this case are few and specific. The sub-biodatabase, its table of bioentries and the rRNA sequence features are the main areas of interest. Feature data specifically rRNAs location, attributes and its direction (strand). After the first data is in place we have the basis to make statistical analyses and compare with the results of the same features from RNAmmer and any other program we wish to see the precision on location found.

To add data to BioSQL database BioPython turns out to be a useful tool,
make it possible to script in the python language for editing the BioSQL data.

Figure 5.1: The parts of the bioSQL schema most relevant, in picture 
biodatabase = sub-biodatabase

5.4 Biopython use

5.4.1 Connect to database

Every time a code needs to communicate (in or out) with the database the communication need to be opened. The user name and password need to be given, and run given into the piece of code under, which create the connection.

```
from BioSQL import BioSeqDatabase

server = BioSeqDatabase.open_database(
    driver="psycopg2",
    user=username,
    passwd=password,
    host=host,
    db=db_name)
```

With an open connection its possible to get, make or delete bioentries. As examples shown under. Note when creating or deleting the change needs to be committed to the database to accuracy happen (last statement)

```
server[biodb_name]
server.new_database(biodb_name)
del server[dbname]
server.commit()
```
5.4.2 Parser

Rather than using Bio.GenBank(), it is now encouraged to use Bio.SeqIO() with the "genbank" format names to parse GenBank files into SeqRecord and SeqFeature objects.

Using Bio.GenBank directly to parse GenBank files is only useful if one wants to obtain GenBank-specific Record objects, which is a much closer representation to the raw file contents than the SeqRecord alternative from the FeatureParser. This created some problem since SeqIO was more applicable and common. So it was the one being used in the first couple of rounds of the building the loadDB code. But in an attempt to get the organism taxonomy of the genome into the bioSQL schema with all its details GenBank parser was the one that needed to be used, it in combination with an iterator.

Classes:
- Iterator Iterate through a file of GenBank entries
- FeatureParser Parse GenBank data in SeqRecord and SeqFeature objects.

 parser = GenBank. FeatureParser()
 records = GenBank. Iterator(open(gbff), parser)

The parser worked but the load function some how did not keep the taxonomy information. So the best solution and the one used for the final program, was to sort into separate sub-biodatabases based on Phylum before reading them into the database.

5.4.3 Seq and SeqRecord

Functions and method used from biopython to make a seqRecord "manually" not just reading with a pre-made parser. Making of a SeqRecord, where the critical part are the Seq for the fasta sequence in it self, and its alphabet (protein or DNA) and the id (an accession number). Then there is the name(version number) and description and many many feature option one example below showing function for adding a feature

```python
from Bio.Alphabet import IUPAC
from Bio.Seq import Seq
from Bio.SeqRecord import SeqRecord

Seq("CATTTGTGAGATCAATAATTGATGCGA", IUPACAmbiguousDNA())
```

```python
rec = SeqRecord(
    S,
    id= acc_id ,
    name= name,
)```
description= "empty",
dbxrefs=[])

product = {"product": attribute}

rec.features.append(
    SeqFeature(
        FeatureLocation(
            start,
            end,
            strand= strand
        ),
        type = f_type,
        qualifiers = product
    )
)

db.load([rec])

The load function at the end is for loading the new record into the database under an specific bioentry. After commit must be used.

5.4.4 Other useful commands

Getting the SeqRecord based on it accession number and extratin a list of all its features, so one can get to all the rRNA in that list.

from Bio.SeqFeature import SeqFeature, FeatureLocation

seq_record = db.lookup(accession= ID)
features = seq_record.features

5.5 The codes

See overview of BioSQL schema and the part we focus on. Flow chart explained

5.5.1 loadDB.py

Starting with the code in loadDB.py, it takes genome data in the form of GenBank files (gbk from NCBI or gbff). It is set up to be able to take several files from a catalog (folder with multiple files), so a group of files can be easily entered together in to the data base under the same run. Then the GenBank files are read one by one, and is parsed into SeqRecord. The SeqRecords taxonomy information is read and sub-biodatabase based on their different phylum are created (and/or added into). Note that the Proteobacteria phylum is split into 5 subgroups. This to sort and store the bioentries
the SeqRecord will loaded up as. Record containing plasmid are eliminated. When all the gbk file has been read turned into SeqRecord sorted and loadet up as bioentries, it time to commit the changes to the database, so they last after the programs done running.

5.5.2 predict.py

Two main parts rRnammerRun and gffLoader
The code in rRnammerRun (should be adjustable for other predicator) runs in this case the RNAmmer annotation search program. gffLoader inputs the resulting data into the database.

predict.py takes a list of phylum and extract one by one the corresponding sub-databases. Each sub-database is then looped through and its bioentries is run in through rRnammerRun. The bioentries being the accession number for each genome. The entries genomes sequences, from the raw data, is extracted in FASTA format and temporarily stored as a txt file. The program RNAmmer is a perl program, so on solution to run it from a python program is to make it "believe" its run from the command line in the terminal. The a read from command line run of RNAmer is setup. Specifiers are added such as the super-kingdom of the input sequence type (bacterial, archaeal or eukaryotic) and the molecule type (5/8, 16/17s, and 23/28s) to search for, and at the and the txt file added. RNAmmer is run and the result can be given as output in three files (gff file, HMM-report file, fasta file). The gff file contains the information needed, namely the rRNA genes such as start, stop, type and direction. So the gff file is the one gotten. The last step of the rRnammerRun.py is taking this information (gff-file) and use gffLoader.py to enter it into the database under rnammer (potentially equivalent program) as the source.

Unfortunately there is no ready made gff parser in BioPython, although it has been worked on. So one had to be made, enter gffLoader. gffLoader takes the bioentrys name, its corresponding gff file and the name of the bioentries sub-database. An new sub-database is created for the new data under a name based on the phylum and the predictor used. The information from the gff-file is then read separated and placed in a SeqRecord with features of rRNA. Then loaded into the new sub-databases.

5.5.3 makePlotData.py

Now all the data, basic raw data and the data resulting from running one or more rRNA finding programs, is all in the same database. The stk function, taking a list of the phylum, get a list each phyla bioentries (the accession number of genomes). With the accession number send into the Stats class to make a genome class. The genome class hold the name of a gen-
ome its unit (5s, 16s and 23) and there start and stop for old and new data. All the genome class for a sub-biodatabase is stored in a list. When all bioentries(genome) under one sub-biodatabases (phylum) is handled, the list get run through makePlotData method. The method run an comparison on the start stop dat for the each of the units, after making sure they are pointing in the same direction and that the amount of “found” units are of same amount as raw count. Finally it put out a txt file with the plot data (unit name start stop and list of difference points).

Example of Tenericutes_plot_data.txt

```
# 5s start
2.0 -3.0 3.0 3.0 -3.0
# 5S stop
-1.0 27.0 -1.0 -1.0 -1.0
# 16s start
-13.0 12.0 7.0 7.0 -10.0
# 16S stop
3.0 -4.0 -9.0 -9.0 0.0
# 23s start
-4.0 3.0 3.0 3.0 -3.0
# 23S stop
-180.0 179.0 -2.0 -2.0 0.0
```

5.5.4 Other minor programs

delet.py

The delet.py function, takes the user name, password and a biodatabase name. It helps remove biodatabase if need be. For example when read in wrong content or names. Also useful for keeping order between runs because, biopythons loader will not allow duplicate of GI number or accession number in same biodatabase.

plot.py

The plot.py uses matplotlib and Seaborn on the separate out data and make violin-plot. These plots show the average distribution of variation between raw data start and stop to the predictors.
Figure 5.2: Flowchart main program
Chapter 6

Results

6.1 Plots

Violinplots of comparison of rRNA location. Each plot represent a different phylum. Only the phylum groups that has 3 or more representation has a plot included since any less data wold not be a good test sample size. The Proteobacteria are divide up in 5 sub group, but only Alphaproteobacteria, Betaproteobacteria and Gammaproteobacteria had 3 or more representations from the raw data.

All the plots are set to show between -100 and 100 on the x-axis representing difference between raw data and predictor results. Note that the x-line has some auto adjustment that change it so plot shows only the most active areas. The small light dot in the middle of the central black line of the plots is the average difference between the predictors and the raw datas locations start and stop.

6.2 Run times

The up-loader and sorter of raw data program loadDB used around 3 hours. The predictor runner and new data up-loader predict

6.3 Observations

There seem to be the beat correspondents between the raw data and the predictor data for the 16s units.

The 5s units are returning quite close results as well.

Table 6.1: Run times

<table>
<thead>
<tr>
<th>Program</th>
<th>Time in hours</th>
</tr>
</thead>
<tbody>
<tr>
<td>loadDB</td>
<td>3.20</td>
</tr>
<tr>
<td>predict</td>
<td>3.00</td>
</tr>
<tr>
<td>makePlotData</td>
<td>1.10</td>
</tr>
</tbody>
</table>
The 23s are not bad but seem to have some more deviation registered.”” Though all units have one or more larger deviation. This can be seen in that the “all data plot” has such wide x-axis.

Figure 6.1: Alphaproteobacteria plot data (n= 14)
Table 6.2: Table over taxonomic category: phylum

<table>
<thead>
<tr>
<th>Phylum</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alphaproteobacteria</td>
<td>14</td>
</tr>
<tr>
<td>Betaproteobacteria</td>
<td>09</td>
</tr>
<tr>
<td>Gammaproteobacteria</td>
<td>34</td>
</tr>
<tr>
<td>Deltaproteobacteria</td>
<td>02</td>
</tr>
<tr>
<td>Epsilonproteobacteria</td>
<td>02</td>
</tr>
<tr>
<td>Actinobacteria</td>
<td>11</td>
</tr>
<tr>
<td>Aquificae</td>
<td>01</td>
</tr>
<tr>
<td>Bacteroidetes</td>
<td>04</td>
</tr>
<tr>
<td>Chlamydiae</td>
<td>03</td>
</tr>
<tr>
<td>Chlorobi</td>
<td>01</td>
</tr>
<tr>
<td>Chloroflexi</td>
<td>01</td>
</tr>
<tr>
<td>Cyanobacteria</td>
<td>03</td>
</tr>
<tr>
<td>Deinococcus-Thermus</td>
<td>03</td>
</tr>
<tr>
<td>Dictyoglomi</td>
<td>01</td>
</tr>
<tr>
<td>Firmicutes</td>
<td>27</td>
</tr>
<tr>
<td>Fusobacteria</td>
<td>01</td>
</tr>
<tr>
<td>Nitrospirae</td>
<td>01</td>
</tr>
<tr>
<td>Planctomycetes</td>
<td>01</td>
</tr>
<tr>
<td>Spirochaetes</td>
<td>03</td>
</tr>
<tr>
<td>Synergistetes</td>
<td>01</td>
</tr>
<tr>
<td>Tenericutes</td>
<td>03</td>
</tr>
<tr>
<td>Thermotogae</td>
<td>01</td>
</tr>
</tbody>
</table>
Figure 6.2: Betaproteobacteria plot data (n= 9)

Figure 6.3: Gammaproteobacteria plot data (n= 34)
Figure 6.4: Actinobacteria plot data (n= 14)

Figure 6.5: Bacteroidetes plot data (n= 4)
Figure 6.6: Chlamydiae plot data (n= 3)

Figure 6.7: Cyanobacteria plot data (n= 3)
Figure 6.8: Firmicutes plot data (n = 27)

Figure 6.9: Spirochaetes plot data (n = 3)
Figure 6.10: Tenericutes plot data (n= 3)

Figure 6.11: All plot data (n= 126)
Chapter 7

Discussion

Here we discuss the project and the development and of the frame work. Have we fulfilled the goals for the project? Which parts worked in and which did not? Thought and comments about the final product. What was achieved, what was not, what could have gone differently.

7.1 The goals

Going over the goals one by one, what work what did not:

- Building a program that take in raw data in Genbank files format (gbk).

This, the loadDB does, being able to read one or several files from a given folder, take the data and add it to into the database as a bioentry for its phylum group (phyla becomes sub-databases). Reading in to several different sub-databases became a work around. This because when reading in the genome, the taxonomy information does not get read in with the rests of the file. Instead the first line in organism gets duplicated over the taxonomy, this happen after parsing into seqRecord then the informations is still there. Its when uploading in it self, a function from the biopython, the data get lost/overwritten, so sorting based on phylum can occur right before uploading.

- Use that data sequences to run new prediction on rRNA location with chosen predictor.

Here rRnammerRun (in predict) goes into the database (then sub-database, then bioentery) and get the previously read in genomes sequences in fasta format. Without problems. Then to use the sequence to run the predictor program (RNAmmer) it need to be stored as a temporary text file, so the RNAmmer program can read it in as an input file at the end of it "read from command line". Ideally this should be possible to avoid since it becomes an extra time constraint, but it does not affect the result.
- Read the new results in to the database.

Here gffLoad (in predict) takes the resulting gff file rRnammerRun produces and makes a seqRecord of it and load it up as under a new (or ad to existing) sub-database, named after predictor and phylum group combined, since thats what was done to its counterpart, and it is the only way of holding on to the taxonomy information. The outcome is that we end up with a lot more sub-databases than first planned, but functionality is not affected.

- Then finally extracts the raw data and the new data and compare its accuracy and presents it in plots.

Extracting the data for comparing, the makePlotData takes one phylum (sub-database name) and get its raw data and its corresponding predictor data. Separates out, unit (5s, 16s and 23) and there start and stop. Then run an comparison on them after making sure they are pointing in the same direction and that the amount of "found" units are of same amount as raw count. Finally it put out a txt file with the plot data ( unit name start stop and list of difference points). The reason for a txt file instead of strait to a plot is mainly the practical problem surrounding the use of the intended plot program (Seaborn). The university computer had a problem installing it correctly only locally. So by obtaining the plot data in txt file the data could easily be moved an run on other devises. But it also useful to see the data obtain in detail, and not only as a pretty plot.

The txt file is then run through plot.py which uses matplotlib and Seaborn separate out the data and make violin-plot. These plots show the average distribution of variation between raw data start and stop to the predictors.

7.2 Results

That 16s unit seem to give better accuracy for RNAmmer corresponds to the fact that the predictor program training data was wides for the 16s unit.

Looks like the big outliers are from Firmicutes just by looking at its plot and the total overview plot.

7.3 Other thoughts

This all with the aim to look at accuracy based on the choice made in the different built of predictor, for example in the use of structure alignment
used in the comparison process.

A problem with all start points of the rRNA features was a output -1 after read in compared to original data. Suspected pythons start from zero range function. It did not however cause the end points to differ, so it was decided to not adjust for it during reading in. But it is something that should be look at and solved.

Shining up the code. Ideally one might want to more or less have it all in one program, where one is asked to enter folder location of raw data then select between predictor on wants to run by just entering its "run from command line" line. Then choose different output data based on phylum or type or other factors. But seeing as the upload process takes around 3 hours both for the raw data and then again for the prediction data, having them as separate running programs gives more time.

Why gff file?
Small with the exact information need and barly any excess. And Barrnap another program looked at the results from also returns in gff format.

Another problem that can occur is a need to separate the seqname in the gff file into GI number and name (accession number plus version number). This was not a problem with output from RNAmmer, so a solution was worked on but did not end up in the final code. The solution is a function that takes ("gi|255534169|ref|NC013062.1") as example and split and sort after ;
- if after "ref""itsanversionnumber
– if anumber followsa"gi" it is an GI number
Chapter 8

Conclusion

The impression we are left with. The frame work was build is should be possible to adjust and us to test the accuracy of rRNA predictor in general. The the use of postgresQL bioSQL work fine, butt biopython as the connetion language had som minor issues that created problem.
Chapter 9

Future Work

What next? One important step that is still to be done is running the program with another predictor. All that is need is download the predictor one wish to test. Make sure it runs. Then edit the "run from command line" prompt in predict.py, adjusting for which type of programing language the predictor uses, and specify a gff output.

It worth not in that in the this thesis has been written a lot has change with the NCBI. As of October 2016 GI identifiers are discontinued in favor of accession numbers. You can still fetch sequences based on their GI, but new sequences are no longer given this identifier. You should instead refer to them by the “Accession number”.
Chapter 10
Appendix

10.1 Abbreviation

GFF    general feature format
OBF    The Open Bioinformatics Foundation
RNA    ribonucleic acid
rRNA   ribosomal RNA
DBMS   Database management system
NCBI   National Center for Biotechnology Information

10.2 Practical details

Choice using biopython(v1.65) on work computer to communicate with a database on a different computer (server at university), setup with postgrSQL(v?). Load in bioSQL schema (v?) on server and work computer

Note

database on a separate server operating system work computer: Ubuntu (linux based) - perl 5.16.3 (for use of rnammer) - eGenix.com mx Extensions - 2.0.x (old) - Have Python 2.7.12 - Installed numpy - Installing BioPython - BioSQL -f http://biopython.org/DIST/ biopython pip install psycopg2

log in direct via terminal(works on uio via ssh) psql -h dbpg-ifi-utv.uio.no -p 5432 -U ailaka -d rnammer tested

You should have a look at postgresql.conf. To allow connections from everywhere you have to set the listen_addresses to: listen_addresses = "*"

the pg_hba.conf should have an entry like that, to allow connection from your network: IPv4 local connections: host all all 127.0.0.1/32 md5 host all all 192.168.190.0/24 md5
Bibliography

