Design and Implementation of a Tool that Supports Evolutionary Analysis of Exon Borders

Sara Farahani

Bachelor's Thesis in Computer Science
at Stockholm University, Sweden, 2018

Stockholm University

Design and Implementation of a Tool that Supports Evolutionary Analysis of Exon Borders

Sara Farahani

Bachelor's Thesis in Computer Science (15 ECTS credits)
Bachelor's Programme in Computer Science
Stockholm University year 2018
Supervisor at the Department of Mathematics was Lars Arvestad
Examiner was Peter LeFanu Lumsdaine

Department of Mathematics
Stockholm University
SE-106 91 Stockholm, Sweden

Abstract

Today, nucleotide data of whole genomes has been sequenced and assembled into databases that are available online. These records become valuable when doing evolutionary studies, and includes biological data such as genes, transcripts and exons. However, some information is not provided in an user-friendly or easily accessible manner, which is the case with exon borders.

The core of this project is to develop a tool that refines the coordinates of exon borders and facilitate the access to them. The project will also involve the management of additional biological data. Among them are gene families, protein coding DNA sequences and transcripts. The project resulted in a relational database in SQLite and three different modules written in Perl. The first script imports biological data from the Ensembl databases using the Ensembl Perl API. The second one imports gene family data from CSV files, and the third one exports exon coordinates together with other nucleotide data in FASTA format.

Design och implementation av ett verktyg som stödjer evolutionär analys av exongränser

Sammanfattning

Idag finns det tillgång till stora mängder genetisk information där arters hela genom har sekvenserats och lagrats i databaser. Uppgifterna finns tillgänglig online och kommer till stor nytta när man gör evolutionära studier. Gener, transkript och exoner är exempel på denna typ av information. Det finns dock problem med vissa data som varken är lättåtkomligt eller användbar. Detta är fallet vid lagringen av exongränser.

Huvudsyftet med detta projekt är således att förbättra tillgängligheten på exonkoordinater och förfina dess data. Projektet involverar även hantering av annan biologisk information, bl.a. genfamiljer, proteinkodande DNA-sekvenser och transkript. Arbetet resulterade i en relationsdatabas i SQLite och tre olika moduler skrivna i Perl. Den första modulen importerar data from Ensembls databaser med hjälp av Ensembl Perl API. Den andra importerar genfamiljer genom att läsa CSV-filer. Den tredje exporterar exonkoordinater tillsammans med annan nukleotid data i FASTA format.

Acknowledgements

This is a Bachelor's Thesis in Computer Science at Stockholm University. The thesis was performed at the Department of Mathematics. My supervisor was Lars Arvestad at Science for Life Laboratory/Dept. of Mathematics, who also was the outsourcer of this project. I would like to thank him for giving me this very educative opportunity and allowing me to have freedom under responsibility. I would also like to thank Monica Bäfverfeldt who took the time to review my report.

Contents

List of Figures

1 Introduction

Due to technical development, researchers have since the 1990s made major progress in identifying genome sequences, i.e. genetic information, of several species (Campbell, Reece, Urry, Cain, Wasserman, Minorsky & Jackson 2015, pp. 55, 287). In order to manage the large amount of data the field of bioinformatics has emerged. Bioinformatics is "the application of computational methods to the storage and analysis of biological data" (Campbell et al. 2015, p. 477). Nowadays we have access to important details, not only to compare genomes of different species, but also to explore biological processes, including evolution (Campbell et al. 2015, pp. 476–477).

Gene families, coding DNA sequences and exon borders are examples of data that are of interest when investigating how species' genetic inheritage has changed over time. This type of data are available in today's online tools, yet the information about exon borders are not always provided in an user-friendly manner. In order to facilitate evolutionary studies involving exon borders and gene families, there is a need for a customized software. Therefore, this project will focus on developing a tool that refines, stores and exchanges this particular information together with other relevant sequence data.

1.1 The Underlying Biology

The following subsections will give a brief introduction to the biological aspects of this project. It will simplify the understanding throughout this report regarding the problem and purpose, as well as the implementation.

1.1.1 Coding Sequence and Protein Sequence

Chromosomes are structures within cells that contains chains of DNA (deoxyribonucleic acid). The DNA strands comprise of nucleotides called adenine (A), thymine (T), cytosine (C), and guanine (G) (Campbell et al. 2015, pp. 53, 371). The nucleotides can form specific coding sequences, called genes, that carries information about protein production (Campbell et al. 2015, pp. 54, 392). One single DNA molecule may include up to thousands of genes (Campbell et al. 2015, p. 53).

Furthermore, a cell also contains proteins, which consist of one or several polypeptides. Every polypeptide is a sequence uniquely built from 20 different amino acids such as Leucine (L) and Valine(V) (Campbell et al. 2015, pp. 125, 127–128). The number of proteins in the human body sums up to tens of thousands, each with a distinct function that is essential to life (Campbell et al. 2015, p. 125).

1.1.2 Transcription and Exons

RNA (Ribonucleic acid) is a molecule constructed by the same nucleotides as DNA, except that thymine is replaced by uracil (U) (Campbell et al. 2015, pp. 392, 394). RNA functions as "The bridge between DNA and protein synthesis" (Campbell et al. 2015, p. 392).

One of the overall processes in the creation of proteins is called transcription. During transcription in eukaryotic cells, information about the protein-coding gene is "rewritten" from DNA to RNA, resulting in a pre-mRNA strand (Campbell et al. 2015, p. 392). The coding segments of a gene, however, are not continuous. There are non-coding sections called introns that are located between other sections called exons, hence the pre-mRNA is structured in a similar way. During RNA processing, the introns in the pre-mRNA are cut away and the exons are then merged into a mRNA sequence, a procedure called RNA splicing (Campbell et al. 2015, pp. 399–400). A demonstration of the latter can be seen in Figure 1.

Later on, the exons are often translated into polypeptides. Still, some regions of the exons are not protein coding, even though they are included in the mRNA (Campbell et al. 2015, p. 399).

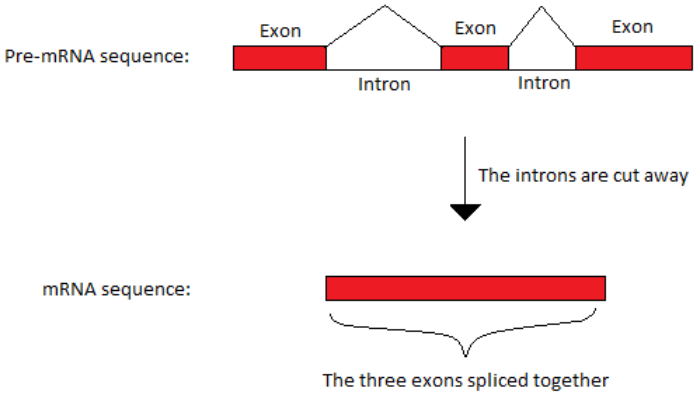

Figure 1: An illustration of RNA splicing.

It is also important to emphasize that a single gene can code for multiple proteins, a possibility given by the existence of introns. Depending on which polypeptide that is being produced, the exons can be defined by different segments during RNA processing, a shift called alternative RNA splicing. As a result, an organism's number of distinct proteins it produces can exceed the number of its genes (Campbell et al. 2015, p. 400).

1.1.3 Gene Families and Gene Alignment

Gene duplication is a recurrent phenomenon in evolution that expands the number of genes in species genomes. Today, it is possible to trace the duplications and divide related genes into gene families (Campbell et al. 2015, p. 535). These genes are called homologous and are represented by DNA sequences derived from the same molecule (Campbell et al. 2015, p. 527).

To evaluate homology among genes, the nucleotides in the DNA molecules are first sequenced. Later, the sequences' comparable segments are aligned. By analysing their similarities and differences it is determined how closely or distantly the species are related to each other (Campbell et al. 2015, p. 528). In practice however, there are tools performing these comparisons together with clustering techniques (Sjöstrand 2013, p. 29). For a graphical description of gene alignment, see Figure 2.

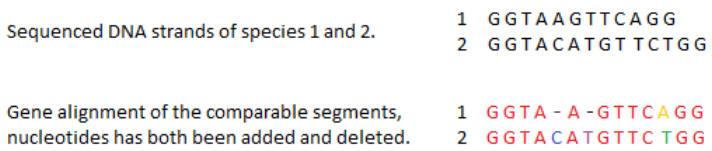

Figure 2: A simple example of gene alignment.

1.2 The Ensembl Project and BioMart

The Ensembl project[1] began developing in 2000 with the primary ambition to sequence the human genome (Birney et al. 2004, p. 925). Nowadays, the system provides biological data from over 70 various species (Ayling et al. 2016, p. 1) and has become "a central hub of genomic information" (Aken et al. 2016, p. D635). Ensembl comprise of a database and genome browser that are regularly updated (Aken et al. 2016, pp. D635–D636). All Ensembl data can be accessed either from a programmatic interface such as Perl API or the web-based interface Ensembl BioMart (Ayling et al. 2016, p. 16). In the Ensembl databases, every object is marked with an Ensembl stable ID which is an unique combination of letters and numbers (Ensembl n.d.).

BioMart is a data mining tool where the user can access data based on selected filters and/or attributes (Ensembl 2017*c*). The filters bounds the query based on information from the user and the attributes regulates the output (Ensembl 2017*d*). For example, the user might have a gene stable ID and choose to retrieve its matching transcript sequences.

[1] `www.ensembl.org`

1.3 Problem and Purpose

Ensembl is a big platform that provides huge volumes of data with varying characteristics. However, some queries results in output that are difficult to conduct further analysis on. The outsourcer of this project performs alignments of coding DNA sequences to define gene families, and is particularly interested in exon borders. This data can be retrieved by using BioMart, but is not optimally generated from an user-friendly perspective. The problems are listed below together with an example output from BioMart[2] in Figure 3.

```
>ENSG00000139618|ENST00000528762|32376670;32379317;32370971;
32375343|32376791;32379495;32371100;32375406

TCATCTGGATTATACATATTTCGCAATGAAAGAGAGGAAGAAAAGGAAGCAGCAAAATAT
GTGGAGGCCCAACAAAAGAGACTAGAAGCCTTATTCACTAAAATTCAGGAGGAATTTGAA
GAACATGAAGTTACTTCCTCCACTGAAGTCTTGAACCCCCCAAAGTCATCCATGAGGGTT
GGAATCAACTTCTGA
```

Figure 3: Example output of a human transcript and its exon coordinates on the chromosome.

- The exon start and end regions are represented by coordinates on the chromosome and not limited to the specific transcript that is being observed.
- The coordinates are not retrieved in sequence order of the exons.
- The start and end regions are listed separately, making it hard to receive a coherent overview.
- The transcript sequence represents the connected exon after RNA splicing. Hence, the exon borders can not be retrieved here either.

The main purpose of this project is to facilitate the procedure of extracting information about exon borders and make the data more usable. As mentioned in section 1.2, it is possible to access the nucleotide data from the Ensembl databases through Perl scripts. Hence, one solution to the output problem described above is to create a tool together with a local database which stores and provides the exon data in a requested way. Since this project is related to evolutionary research, it is also within the scope to manage other biological information and to enable the data to interact with other tools traditionally used in evolutionary studies.

[2]Note that transcripts are represented by DNA nucleotides in Ensembl.

1.4 Limitations

The project will only include an implemented solution that is exclusively compatible with the Ensembl databases. The local database will be a relational database implemented in SQLite. It will only manage the following biological data: species, genes, transcripts, exons, gene families and proteins.

The tool will import the greater part of the nucleotide information to the database by using Ensembl Perl API. It will also handle comma-separated values (CSV) files in text format as input, containing gene families defined by the user. Finally, the tool will write text files in FASTA format including exon borders and nucleotide data from selected families.

The source code will be uploaded on www.github.com. A graphical user interface will not be implemented as the command line will be used to interact with the system.

2 Design and Implementation

Figure 4 shows an overview of the different components involved in this project. During the development, three scripts and one database was implemented. The right bin symbolises all of the Ensembl databases where the script `import_from_ensembl` extracts nucleotide data from, and inserts it to the local database. The `insert_gene_families` script reads a CSV file provided by the user and stores the records in the local database as well. The functionality of the third script, `write_gene_families`, is to export gene family data including exon borders in FASTA format. The following subsections will present a more detailed description of the implementation of each part. All source codes can be found at `www.github.com/SaraFarahani/exon_borders_tool`.

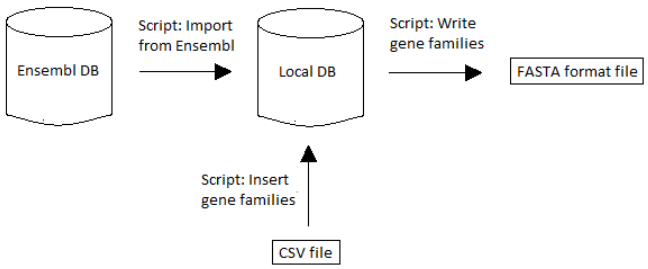

Figure 4: An overview of the tool. The three arrows illustrates the data exchange performed by the modules of this project.

2.1 Developing a Nucleotide Database

A relational database was developed to meet the outsourcer's need to store large amounts of data. The database structure is based on biological aspects, described in section 1.1, and Ensembl's own database schema[3] . After completing the structure, the database was implemented in SQLite. The following sections specifies the approach to these processes. Additionally, for further reading on the database concepts mentioned below, see Harrington (2016).

2.1.1 The Database Schema

As seen in Figure 5, the database schema consists of eight tables, their attributes, and the relationships among the tables. The underscored attributes

[3] `www.ensembl.org/info/docs/api/core/core_schema.html`

are the primary keys of each table. The arrows linking the tables either represent one-to-one or one-to-many relationships. They also clarify the foreign key constraint that are used to connect two tables to each other. In the one-to-many relationships it is the "many" that holds the "one" key, for the other relationship this was chosen at random. In cases where a many-to-many relationship exists, an additional table has been added. It consists only of the two foreign keys from the tables sharing the relationship, e.g. `gene_family`.

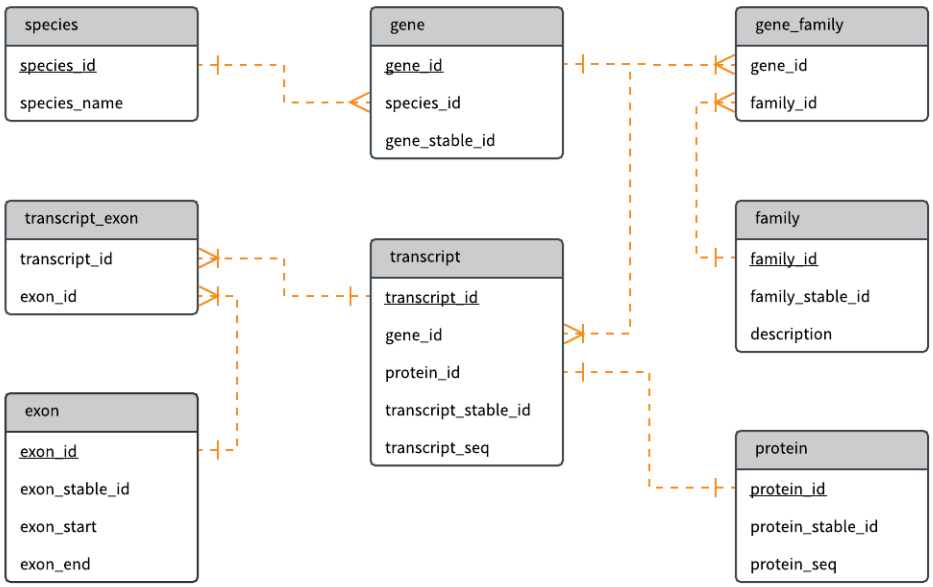

Figure 5: The database schema

The normalization degree of the database structure has been analysed up to Boyce-Codd normal form (BCNF). The reasoning in this context are based on definitions that can be found in Harrington (2016).

- The first normal form is reached since each attribute in this database is limited to having only one value in every row.
- The second normal form is also meet. All of the non-key attributes are functionally dependent on the entire primary key.
- The requirements for third normal form regarding transitive dependencies are also satisfied. In this database, the transitive dependencies is recognized by seeing the unique stable_ids as second determinants. In this case however, transitivity is allowed since these stable_ids are candidate keys.

- Every determinant is a candidate key, hence the database is in BCNF.

2.1.2 Implementing the Database Using SQLite

The outsourcer of this project requested that the relational database management system should be SQLite. It is a broadly used SQL database engine that comes with many advantages in terms of user-friendliness. SQLite does not require a separate server to operate on and its code is in the public domain. SQLite is said to be a very reliable system and has a maximum database size of 140 terabytes (SQLite n.d.).

The database was developed in SQLite version 3.18.0. The tables were created using regular SQL queries. Besides from implementing primary keys, the queries also regulates other constraints, such as unique, not null and referential integrity to foreign keys. An example query is shown below:

```
1 CREATE TABLE gene(gene_id integer PRIMARY KEY NOT NULL,
    species_id integer NOT NULL, gene_stable_id varchar(255)
    UNIQUE NOT NULL, FOREIGN KEY(species_id) references species
    (species_id));
```

Furthermore, the attribute values `species_name` and `*_stable_id`[4] were set to unique to avoid duplicates. An exception is the `exon_stable_id`, which is allowed to occur more than once (for an explanation of why, see last part of section 2.2.1). The only column that allows null entries is `description` in the `family` table. Furthermore, the constraint "on delete cascade" has been applied to the `gene_family` table, meaning that if the user manually deletes a family, the corresponding records in the `gene_family` table will also be deleted.

2.2 Database Interaction With Perl Scripts

The first step after developing the database was to import records from the Ensembl genome databases. It is recommended to use the Ensembl Perl API if large volumes of records are to be extracted from their databases (Ensembl 2017*a*). Hence, the import script was written in Perl, which also is compatible with SQLite. By convenience the same language was used to develop the other two components of the tool.

All of the scripts use the Perl module DBI to connect with the local database and perform data exchange. While making entries to the database all primary keys are automatically generated. The subsequent sections provide a detailed description of the three self-contained programmes that was created in this project, explaining both how and why they were built.

[4]The asterix stands for gene, transcript, exon etc.

2.2.1 Importing Nucleotide Data Using The Ensembl Perl API

Ensembl has stored their records in a publicly-accessible MySQL server. Data is retrieved by using the Ensembl Perl API, which enables records to be accessed without knowing the database schema (Ensembl 2017*a*). The Perl API is object oriented and includes a vast amount of classes, object adaptors and methods. Ensembl has provided a brief introduction of how to use some of them[5]. This was used to familiarize with the syntax. During the development relevant methods was found by the means of the API documentation[6]. The names of the methods from the Ensembl Perl API are self-explanatory, consequently there will be no detailed description of them here. The remaining part of this section will address the coding part of this module.

2.2.1.1 Retrieving Translatable Transcripts

As described in the API tutorial, the registry module is imported in order to connect with the Ensembl databases. By using the registry, an object adapter and a fetch method, all transcripts for a chosen species is retrieved from the core database (Ensembl 2017*b*). The `fetch_all()` method results in a list reference and the content is dereferenced into an array. Since only the protein coding transcripts are of interest, the non-coding transcripts are sorted out.

```
1 use Bio::EnsEMBL::Registry;
2 my $transcript_adaptor = $registry->get_adaptor($species, '
    Core', 'Transcript');
3 my $transclones_ref = $transcript_adaptor->fetch_all('clone');
4 my @transclones = @{$transclones_ref};
5
6 my @translateable_transcripts=();
7 while ( my $transcript = shift @transclones) {
8   my $translateable = $transcript->translateable_seq();
9   if ($translateable ne ''){
10    push (@translateable_transcripts, $transcript);
11  }
12 }
```

2.2.1.2 The Main Function

The array with translatable transcripts is sent to the subroutine `main()`. This function manages the systematics of all entries to the local database. Every table in the database has its own subroutine, these are called by the

[5] `www.ensembl.org/info/docs/api/core/core_tutorial.html0`
[6] `www.ensembl.org/info/docs/Doxygen/core_api/index.html`

main function. It also ensures that correct input values are included in each call.

2.2.1.3 Insert Functions and Help Functions

The majority of the code consists of insert functions and get functions (help functions). The insert functions (one for each table) handle the entries to the local database. These subroutines always receives a transcript object as input. The transcript object is used to retrieve nucleotide data from the Ensembl databases, e.g. a species name or a gene stable id. Depending on the structure of the table, some insert functions have additional input values. These are foreign keys that has been returned from the other insert functions and passed forward by the main function. Furthermore, the code contains several help functions. Each of them has a specific assignment retrieval, e.g. to fetch a gene id or a transcript sequence.

Moving on the technical parts of the database entries, the subsequent three sections will describe the most importing aspects of the code. For the complete code, see `www.github.com/SaraFarahani/exon_borders_tool/`.

2.2.1.4 Avoiding Duplicates in the Gene Table

To manage the `gene` table, it is required to retrieve the `gene_stable_id` of each transcript. Since a gene can have several transcripts, the same `gene_stable_id` might appear more than once. If the SQL query tries to insert an already existing stable id, the programme will break because duplicates are not allowed in the `gene` table. To avoid this, all insertions to the gene table begins with "INSERT OR IGNORE". The statement ensures that the entry is performed only if the `gene_stable_id` does not already exist.

2.2.1.5 Constructing the Transcript Sequence

In the `transcript` table, a transcript sequence is constituted by its corresponding exons. The sequence is formed by the help function `get_transcript_seq()`. The subroutine retrieves a list of ordered exon references and dereference it into an array. A while-loop iterates through the array where the exon's sequence is fetched and pushed into another array. This array is later joined in itself, forming the complete transcript sequence.

```
sub get_transcript_seq{
  my $transcript = shift;
  my $exons_ref=$transcript->get_all_Exons();
  my @exons_array = @{$exons_ref};
  my @transcript_array= ();
  while(my $exon_ref = shift @exons_array){
    my $exon_seq = $exon_ref->seq()->seq();
    push (@transcript_array, $exon_seq)
  }
  my $transcript_seq = join("", @transcript_array);
  return $transcript_seq;
}
```

2.2.1.6 Converting the Exon Coordinates

As mentioned in section 1.3, the exon borders are represented by coordinates on the chromosome. The goal is to convert them into transcript coordinates. The exons occur in order when they are retrieved from the Ensembl databases. This means that by determining the length of each exon, it is possible to transform the chromosome coordinates to transcript coordinates.

In the first exon of every transcript, the attribute values `exon_start` is set to 1 and `exon_end` is set to the sequence length+1. In the process for the remaining exons, the subroutine `get_coordinates()` uses another help function to fetch the last inserted `exon_id` before assigning the start and end positions as follows:

```
sub get_coordinates{
  my $exon = shift;
  my $previous_exon_end = get_last_exon_end();
  my $new_exon_start = ($previous_exon_end+1);
  my $old_exon_start = $exon->seq_region_start();
  my $old_exon_end = $exon->seq_region_end();
  my $new_exon_end = $new_exon_start + ($old_exon_end-
    $old_exon_start);
  my @coordinates = ($new_exon_start, $new_exon_end);
  return @coordinates;
}
```

Similar to the relationship between transcripts and genes, an exon can be included in several transcripts. In contrast to the earlier situation, the exon table allows duplicates of `exon_stable_ids`. The reason for this is that even though an exon always has the same chromosome coordinates, it might not be assigned with the same transcripts coordinates. Here is an example:

- In one transcript, exon A is followed by exon C. In another transcript, exon B is also followed by exon C.
- The chromosome sequence regions are A: 1–5, B: 1–10, C: 15–20.

- The script will then set the transcript start and end positions to A,C: 1–5, 6–11 and B,C: 1–10,11–16.

It is now clear that an exon's transcript coordinates partially depends on the sequence region of the former one, and that the same exon might be given different transcript coordinates. Still, the transcript regions are in the same scale as in the chromosome but has been shifted to become more coherent. The duplicates will not cause any problems since the `transcript_exon` table keeps track of the `exon_ids` together with its corresponding `transcript_ids`.

2.2.2 Importing Gene Families by Reading a CSV File

Another function requested by the outsourcer was the possibility to store records about gene families. As the outsourcer uses standard tools to perform gene alignments, new family definitions are formed. This information is summarized in a CSV file. In this case, the file contains rows of comma-separated gene ids, where each line represents one gene family. In order to assemble data from different gene alignments, another script was written in Perl.

As the user starts the script, the programme will ask the user to enter family stable ids and family descriptions to the new families. The subroutine `check_arrays()` will ensure that the number of family stable ids and family descriptions matches the number of families in the CSV file. For each row in the file, the function `insert_family()` performs entries to the `family` table and call the `insert_gene_family()` function. The latter inserts records to the `gene_family` table by iterating through the row of gene ids.

2.2.3 Exporting Exon Borders in FASTA Format

Apart from storing records of biological data, the outsourcer is also interested in using the data in other contexts. As described in section 1.3, the core of this project is to generate exon borders in a more accessible and useful manner. More specifically, it was requested that the tool would produce a text file containing nucleotide data in FASTA format. FASTA format is a certain syntax used to transform biological data into a text-based format. Mainly, it begins with the symbol ">" followed by one descriptive line and ends with lines of sequence data (University of Michigan n.d.).

To retrieve records from the database in FASTA format, a third module was created. When starting the program, the user enters an optional number of family stable ids which corresponds to the gene families that should be included in the output file. The module contains a `main()` function which builds the FASTA formatted text file by calling a number of help functions. The help functions provide the `main()` function with requested records from the database.

3 Results

The following sections will demonstrate the outcome from running the different scripts. Example data of input and output, as well as records from the local database will be presented. The execution times will also be included.

3.1 Importing Nucleotide Data

The script, `import_from_ensembl`, that imports records from the Ensembl databases has been run for several species. Figure 6 and 7 show selected database output from the `species` and `gene` tables. In this case the species were first *armadillo* and later *Saccharomyces cerevisiae*.

species_id	species_name
1	dasypus_novemcinctus
2	saccharomyces_cerevisiae

Figure 6: Sample output from the species table.

gene_id	species_id	gene_stable_id
22709	1	ENSDNOG00000046561
22710	1	ENSDNOG00000040274
22711	1	ENSDNOG00000048587
22712	2	YHR055C
22713	2	YPR161C
22714	2	YOL138C

Figure 7: Sample output from the gene table.

The output from the `exon` table seen below shows how the exon borders are displayed in the database. Figure 8 illustrates the exon coordinates for one *armadillo* transcript (the transcript constitutes of several exons). Figure 9 visualizes the exon coordinates for several *Saccharomyces cerevisiae* transcripts (each transcript constitutes of one exon only).

To ensure that the data was stored correctly, randomly chosen records were manually traced among the tables and compared to the records in BioMart and Ensembl's data display[7]. No deviations were found.

[7] `http://www.ensembl.org/info/website/gallery.html`

exon_id	exon_stable_id	exon_start	exon_end
87	ENSDNOE00000384048	1	6195
88	ENSDNOE00000095721	6196	6329
89	ENSDNOE00000365959	6330	6505
90	ENSDNOE00000095752	6506	6607
91	ENSDNOE00000095765	6608	7113
92	ENSDNOE00000425443	7114	9013

Figure 8: Sample output from the exon table, one armadillo transcript.

exon_id	exon_stable_id	exon_start	exon_end
243388	YHR055C.1	1	186
243389	YPR161C.1	1	1974
243390	YOL138C.1	1	4026
243391	YDR395W.1	1	2835

Figure 9: Sample output from the exon table, several Saccharomyces cerevisiae transcripts.

Table 1 summarizes the approximate execution times of the script with the two species. The total execution time was 22 and 2 hours for *armadillo* and *Saccharomyces cerevisiae* respectively.

Table 1: Execution times for the two test species.

Species	Retrieve all transcripts	Sort out non-translatable transcripts	Retrieve and insert specific data	Number of transcripts/ exons
armadillo	200 s	2 h	20 h	26551/243387
Saccharomyces cerevisiae	12 s	0.5 h	1.5 h	6692/7050

3.2 Inserting Gene Family Data

Before running the module `import_gene_families`, a CSV file was created. The file contained arbitrary selected gene stable ids that were divided into three families shown in Figure 10.

When running the script, family stable ids and descriptions was entered. The programme inserted the records to the `family` and `gene_family` table seen in Figure 11 and 12. Note that the same gene can be included in several families depending on what the family definition is.

```
ENSDNOG00000032971,ENSDNOG00000030653
ENSDNOG00000049452,ENSDNOG00000031262,YHR055C
ENSDNOG00000030653,YOR280C,YKL025C
```

Figure 10: Input of gene families in the CSV file where each row constitutes a gene family.

family_id	family_stable_id	description
1	FAM1	Same first nucleotides
2	FAM2	Same last nucleotides
3	FAM3	Same middle nucleotides

Figure 11: Output from the family table.

gene_id	family_id
22632	1
22663	1
79	2
22635	2
22712	2
22663	3
22728	3
22731	3

Figure 12: Output from the gene_family table.

The execution time for the three families was 1 second. Another test case was performed with an input file including ten families, each containing ten gene ids. The time result was 11 seconds. The `check_arrays()` call took less than 1 second and `insert_family()` together with `insert_gene_family` took about 10 seconds.

3.3 Exporting Coordinates of Exon Borders

In the last module, the three gene families from the former script were chosen. The FASTA formatted text file includes `family_stable_id`, `gene_stable_id`, `exon_start`, `exon_end` and `transcript_seq`. In Figure 13 the descriptive lines actually corresponds to one row in the text file. Also notice that the last part of the sequences has been cut away.

Arbitrary parts from the output file were reviewed by using the database records as reference. The execution time was less 1 second. The result for retrieving data from 10 different families, each containing 10 genes, was 1 second.

```
>FAM1|ENSDNOG00000030653|ENSDNOT00000030702|1;450;
ATGGCTTCTTCTGATATCCAGGTCAAAGAACTGGAGAAGCGTGCCTCAGGCCAGGCCTTTGAGCTTATA
CTCAGCCCTCGGTCAAAAGAAACTGTCCCAGAATTTCCCCTTTCCCCTCCAAAGAAGAAGGATCTTTCC
CTGGAGGAAATTCAGAAGAAGTTAGAAGCTGCAGAAGAAAGACGCAAGTCCCATGAAGCTGAGGTCTTG
>FAM2|ENSDNOG00000049452|ENSDNOT00000046154|1;169;170;321;322;406;407
;594;595;722;723;834;835;5763;
GCGGGCGGCGTCGGGGCGCGGACTCGGCGGGAGCGGCGCGCACTGTCCGGCGGGGACCCCCGCGCCTCG
GCGCGCCATGCCCCAGACCGTGGTGCTCCCGGGCCCCGCGCCCTGGGGCTTCAGGCTCTCGGGGGGCAC
GGACTTCAGCCAGCCTCTGGTCATCACCAGGATTACTCCAGGAAGCAAGGCGGCAGCTGCCAACCTGAG
TCCTGGCGATGTCATCCTGGCTATCGATGGCTACAGTACAGAGTTCATGACTCATGCTGATGCACAGGA
```

Figure 13: Sample output from the FASTA file.

4 Conclusion

Given the results above, the outsourcer's needs in this context has been met. The completed tool imports, stores and exports the requested information. The main purpose of this project was to facilitate the access to exon borders. This project has provided one possible solution to improve data display and data availability.

The script that interacts with the Ensembl databases retrieves the nucleotide data and modifies the exon coordinates before entering the information to the local database. As demonstrated in the example outputs above, the records are stored in a well-structured and easily accessible manner. This also applies to the result from the insert script which manages gene families. The greatest progress is noticed in the examples from the FASTA formatted file. Compared to the output from BioMart, the exon coordinates are now based on the transcript sequence, in subsequent order and easier to interpret.

However, there exist a concern regarding the execution time in the nucleotide import script. It is very time consuming in some cases. As presented in Table 1, the total time was 22 respectively 2 hours. The table reveals that the process of retrieving specific data from the Ensembl databases and perform entries to the database stand for the larger part of the execution time.

5 Discussion

When working through this project, the input and output was provided by the outsourcer and the focus has been on finding methods to manage them. However, managing the data seems to be time consuming. Table 1 shows that the execution time can vary a lot among different species. In these two cases, there also exist a great difference in their genomes. The *armadillo* has more genes than the *Saccharomyces cerevisiae*. Additionally, the *armadillo* has more transcripts constituting of several exons compared to the *Saccharomyces cerevisiae*. To put this into programme perspective, the genes consist of one or several transcripts which in turn are built by one or several exons. The code involves iterations through all of these transcripts and exons, as well as repeated database exchange using every single one of the objects.

It has not been investigated whether a programme with another approach would be less time consuming, or if the problem depends on the machine or the SQLite file system. This could be evaluated in future research. However, time efficiency was not a requirement from the outsourcer. The imports from Ensembl databases are one time procedures that will result in stored data that is valuable, both in this context and in future developments.

References

Aken, B. L., Achuthan, P., Akanni, W., Amode, M. R., Bernsdorff, F., Bhai, J., Billis, K., Carvalho-Silva, D., Cummins, C., Clapham, P. et al. (2016), 'Ensembl 2017', *Nucleic Acids Research* **45**, D635–D642.

Ayling, S., Aken, B. L., Barrell, D., Clarke, L., Curwen, V., Fairley, S., Banet, J. F., Billis, K., Girón, C. G., Hourlier, T. et al. (2016), 'The Ensembl Gene Annotation System', *Database* **2016**(baw093), 1–19.

Birney, E., Andrews, T. D., Bevan, P., Caccamo, M., Chen, Y., Clarke, L., Coates, G., Cuff, J., Curwen, V., Cutts, T. et al. (2004), 'An overview of Ensembl', *Genome research* **14**(5), 925–928.

Campbell, N. A., Reece, J. B., Urry, L. A., Cain, M. L., Wasserman, S. A., Minorsky, P. V. & Jackson, R. B. (2015), *Biology: A Global Approach*, Harlow: Pearson Education Limited.

Ensembl (2017*a*), 'Accessing Ensembl Data', `https://www.ensembl.org/info/data/index.html`. Accessed: 31 Oct 2017.

Ensembl (2017*b*), 'Ensembl Core Api Tutorial', `http://www.ensembl.org/info/docs/api/core/core_tutorial.html`. Accessed: 31 Oct 2017.

Ensembl (2017*c*), 'Extracting data with biomart', `http://www.ensembl.org/info/data/biomart/index.html`. Accessed: 15 Oct 2017.

Ensembl (2017*d*), 'How to use biomart', `http://www.ensembl.org/info/data/biomart/how_to_use_biomart.html`. Accessed: 15 Oct 2017.

Ensembl (n.d.), 'I have an Ensembl Id, what can i tell about it from the ID?', `http://www.ensembl.org/Help/Faq?id=488`. Accessed:15 Oct 2017.

Harrington, J. L. (2016), *Relational database design and implementation*, Cambridge: Morgan Kaufmann.

Sjöstrand, J. (2013), *Reconciling Gene Family Evolution And Species Evolution*, PhD thesis, Stockholm: Dept.of Numerical Analysis and Computer Science, Stockholm University.

SQLite (n.d.), 'About sqlite', `https://www.sqlite.org/about.html`. Accessed: 31 Oct 2017.

University of Michigan (n.d.), 'What is FASTA format?', `https://zhanglab.ccmb.med.umich.edu/FASTA/`. Accessed: 31 Oct 2017.