ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE

FAKULTA BIOMEDICÍNSKÉHO INŽENÝRSTVÍ

Katedra biomedicínské informatiky

NSAT

Nucleotide Sequence Analysis Toolkit

Bachelor`s thesis

Study programme:	Biomedicínská a klinická technika
Field of study:	Biomedicínská informatika
Bachelor`s thesis author:	Matěj Nemec
Bachelor`s thesis advisor:	Ing. Jan Tesař

Kladno 2018

Katedra biomedicínské informatiky Akademický rok: 2017/2018

Zadání bakalářské práce

Student:	**Matěj Nemec**
Obor:	Biomedicínská informatika
Téma:	**Aplikace pro výpočet genomických parametrů**
Téma anglicky:	Application for determination of genomic parameters

Zásady pro vypracování:

Cílem bakalářské práce je vytvořit aplikaci pro srovnávání genetických sekvencí, pomocí výpočtu genomických parametrů ANIb (Average Nucleotide Identity), Tetra (tetranucleotide signature correlation index) a GC pairs (počet párů tvořených bázemi Guanin a Cytosin v sekvenci). Sekvence bude možné porovnávat vzájemně mezi sebou, dále určit libovolné target sety (cílových sekvencí) a query (sekvencí, které chceme srovnat s target setem). Výstupem aplikace bude grafická reprezentace získaných dat v podobě dendrogramu a textová reprezentace v podobě matice výsledků exportovatelné do formátu .csv.

Seznam odborné literatury:

[1] Christian Nagel, Bill Evjen, Jay Glynn, C# 2008 programuje profesiálně, ed. 1, Computer Press, 2009, ISBN 978-80-251-2401-7
[2] Phillip Compeau, Bioinformatics Algorithms An Active Learning Approach, ed. 1, Active Learning Publishers LLC, 2014, 384 s., ISBN 9780990374619
[3] Fatima Cvrčková, Úvod do praktické bioinformatiky, ed. 1, Academia, 2006, ISBN 80-200-1360-1

Zadání platné do:	20.09.2019
Vedoucí:	Ing. Jan Tesař

[illegible] [illegible]

....

vedoucí katedry / pracoviště děkan

V Kladně dne 19.02.2018

PROHLÁŠENÍ

Prohlašuji, že jsem bakalářskou práci s názvem „Název práce“ vypracoval/a samostatně a použil/a k tomu úplný výčet citací použitých pramenů, které uvádím v seznamu přiloženém k diplomové práci.

Nemám závažný důvod proti užití tohoto školního díla ve smyslu § 60 Zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon), ve znění pozdějších předpisů.

V Kladně 17.8.2018 …...…………...………………...

Matěj Nemec

ACKNOWLEDGEMENTS

I would like to thank anyone and everyone who helped me to make this project a reality whether it was directly or indirectly.

I want to thank my family for supporting me during my studies and esepcially my father for providing valued feedback. Furthemore my friends Martin Španěl and Ondřej Košut, who both are great programmers and intelligent people deserve a special acknowledgement for not laughing at me when I get stuck on a rather trivial problem and instead offering advice.

I would also like to thank the staff of LBG for providing support and advice with my thesis as well as the SZÚ instituition for giving me space a means to work on it.

My teachers from Faculty of Biomedical Engineering at CTU deserve thanks for teaching me a lot of the thing needed to even attempt such a project and of course a special thanks goes to my thesis advisor Ing. Jan Tesař for having the nerves to deal with me, which is no easy feat.

And finally I would like to thank caffeine which, while it might not be the healthiest thing on earth made finishing this thesis on time possible.

ABSTRACT

NSAT: Nucleotide Sequence Analysis Tool

This thesis is focused on in silico comparison of genomes for the purposes of bacterial systematics. Its goal was to create a user-friendly, offline software package that provides large-scale analysis of complete or draft whole-genome sequences. We examined most common methods and genomic parameters used for these purposes in order to design the application named NSAT. The selected algorithms were reviewed in detail and broken down to the individual steps and mathematical formulas. The NSAT program uses the Average Nucleotide Identity, Tetranucleotide frequency correlation and GC content in percentage parameters. It was implemented in the C# programming language using .NET framework library and compiled to run on the Microsoft Windows platform. NSAT was designed with a simple graphical user interface and can be operated by non-bioinformatician personnel. Its functionality and result accuracy were tested using a set of reference sequences and found satisfactory.

Keywords

ANIb, TETRA, DNA sequence comparison, in silico, bioinformatics

Table of contents

List of symbols and abbreviations

List of symbols

Symbol	Unit	Meaning
T_m	$°C$	Melting point at which the DNA strands separate
ΔT_m	$°C$	Difference between the melting points

List of abbreviations

Abbreviation	Meaning
rRNA	Ribosomal RNA
DDH	DNA-to-DNA hybridization
ANI	Average Nucleotide Identity
ANIb	Average Nucleotide Identity based on the BLAST algorithm
ANIm	Average Nucleotide Identity based on the MUMmer algorithm
%GC	Content of G a C bases in the examined sequence in percent
BLAST	Basic Local Alignment Search Tool
XNA	Xeno nucleic acid
A	Adenine
T	Thymine
G	Guanine
C	Cytosine
IUPAC	International Union of Pure and Applied Chemistry
IUB	International Union of Biochemistry
NCBI	National Center for Biotechnology Information
BLOSUM	Blocks Substition Matrix
PAM	Point Accepted Mutation
HSP	High Scoring Pair (BLAST context)

b	DNA base
Kb	Kilo-base – 1000 DNA bases
TETRA	Tetranucleotide frequency correlation coefficient
UI	User Interface
GUI	Graphical User Interface
SZU	Czech National Health Institute (Státní Zdravotní Ústav)
LBG	Laboratory of Bacterial Genetics at SZU
UPGMA	Unweighted Pair Group Method with Arithmetic Mean
IDE	Integrated Development Enviroment
r	Pearson`s correlation coefficient
WPF	Windows Presentation Foundation
Newick	New Hampshire tree format
UML	Universal Modelling Language
BLASTN	Nucleotide BLAST
AAI	Average Amino Acid Identity
MiSI	Miscrobial Species Identifier
MIT	Massachusetts Institute of Technology
NSAT	Nucleotide Sequence Analysis Tool
I/O	Input/Output

1 Introduction

Systematics of bacteria has always depended on developments in other scientific disciplines. Following the discovery of its structure in the early 1950s, DNA has been gradually established as the most relevant marker used for comparative studies in bacterial taxonomy. Its first application was the determination of the guanine-cytosine content (%GC) of bacterial chromosomes, which was followed by the introduction of rRNA-DNA and DNA-DNA hybridization assays, and later of the sequence analysis of essential, house-keeping genes encoding 16S rRNA or proteins.

Quantification of genetic relatedness by DNA-DNA hybridization (DDH) became a basic tool for the classification of bacteria at the species level in the 1980s. In their Report of the Ad Hoc Committee on Reconciliation of Approaches to Bacterial Systematics (Moore, 1987), postulated that *the phylogenetic definition of a species generally would include strains with approximately 70% or greater DNA-DNA relatedness and with 5°C or less* ΔT_m (difference in melting temperature between hybrid mixture and control). Skackebrandt and Goebel (STACKEBRANDT, 1994) then suggested that 16S rRNA sequence similarity of less than 97% between bacterial organisms indicates that they represent different species, whereas at 97% or higher 16S rRNA similarity, DDH must be used to determine whether they belong to one species. Thus, the combination of DDH and 16S rRNA analysis has been established as a gold standard for the phylogenetic classification of bacteria and become an essential methodical component in descriptions of novel species.

Despite the crucial role of DDH in the developments of modern taxonomy, this technique suffers from methodical and methodological limitations that hamper a more effective and accurate extraction of the taxonomic information embedded in genomic DNA. DDH is technically demanding while diverse methods developed to perform it can yield different results, especially for lower hybridization values. However, the main drawback of this technique is its comparative nature. As similarity (DNA-DNA relatedness) values are the results of laboratory experiments, no incremental databases can be built. This contrasts with the situation of sequence analysis, when similarity values are obtained through computer-based comparison of digitized DNA sequences.

In the aforementioned article, Wayne wrote that *there was general agreement that the complete DNA sequence would be the reference standard to determine phylogeny and that phylogeny should determine taxonomy* (Moore, 1987). Although this claim was not practically achievable until the dawn of the new millennium, the recent introduction and development of high-through-put, next-generation sequencing techniques have completely changed the situation. Nowadays, obtaining a nearly complete bacterial genome sequence is almost a routine procedure while techniques for the determination of a complete genome sequence are easily available.

The high speed and low cost of draft genome sequencing have motivated taxonomists to search for in silico methods of genome comparison that would eliminate the drawbacks associated with conventional DDH. Several parameters have been developed to compare genome sequences for species circumscription, of which the average nucleotide identity(ANI) (Konstantinidis, 2005a) has been most commonly applied in taxonomic studies. ANI score calculation can be achieved using two different algorithms. One is ANI based on BLAST searches of ca. 1 kb genome fragments (that mirrors the fragmentation of genomic DNA during conventional DDH experiments) against a target genome, ANIb (Klappenbach, 2007), whereas the other one is based on the MUMmer algorithm, ANIm (Richter, 2009), which is of higher speed and does not require the artificial generation of 1 kb fragments. Although both algorithms give nearly identical values in the high identity range (90–100%), their results diverge for less similar genomes and in these cases the ANIb algorithm appears to produce more accurate results (Richter, 2009). Of the two ANI variants, ANIb is currently considered a standard (Rosselló-Móra, 2015).

In the last few years, several web applications allowing for the determination of ANI and some additional parameters have been released. The first was the JSpecies software (Jspecies, 2009) written in the Java programming language (Richter, 2009) and runs locally on Linux or Microsoft Windows. Even though it is able to calculate ANIb, ANIm, tetranucleotide frequency and %GC and provides a polished graphical interface (e.g. to show the distributions of fragments according to ANI values), it suffers from several flaws. These include the inability to run MUMmer algorithm (and therefore calculate ANIm) under Windows, incorrect calculation of %GC and a rather low number of sequences (less than 20) for which mutual ANIb values can be factually calculated under Windows. A newer, online version of the program termed JSpeciesWS (JspeciesWS, 2015) (Rosselló-Móra, 2015) has solved some of these problems but still enables only a limited maximum number (15) of sequences to be compared, which precludes the comparison of larger genome sets for comprehensive taxonomic studies. Although other online applications that offer ANI calculation are currently available, they also suffer from the capacity problem.

My motivation for this thesis arose from discussions with the scientific team of the Laboratory of Bacterial Genetics (National Institute of Public Health, Prague). The team members drew my attention to the practical limits of the presently available applications in the situation when the number of genomes to be compared grows quickly. It became apparent that there was a need for an offline bioinformatics package capable of computing standard genomic parameters for large sets of genomes, which would be flexible enough to set important variables or select genomes to be compared as well as provide a graphical representation of the overall similarity between sequences based on cluster analysis. Although none of these components is original, the planned software package has a clearly defined purpose and promises to fill an important methodical gap.

The goal of this thesis is to design, implement, test, and release a bioinformatics toolkit with graphic user interface, the main functionality of this application lies in comparing genomic parameters between the nucleotide sequences of bacterial whole genomes. The toolkit will be written in C# programming language as a Microsoft Windows application supporting Windows 7 and newer, requiring an x86 architecture CPU to run the program.

2 Current state

2.1 DNA and its representation

2.1.1 DNA molecule

DNA (deoxyribonucleic acid) is a molecule present in every known living cellular organism. Initial discovery of the molecule dates back to late 1860s but its significance, function and structure was understood almost a century later. Most notably, model of its double helix structure was first introduced by Watson and Crick in 1953 (A. Pray, 2008). We could argue that DNA is a prerequisite for life as we know it, although there are successful attempts to create a synthetic DNA-like XNA molecule, which differs from DNA by replacing deoxyribose in the backbone with a different sugar molecule, suggesting that there is a possibility of XNA based life (Edited by Markus Schmidt., 2012).

DNA serves as a blueprint for the entirety of the organism in question, guiding its development from single cell to its final form. Its structure consists of four different nitrogen bases: adenine (A), cytosine (C), guanine (G) and thymine (T), which are connected by a sugar (deoxyribose)-phosphate backbone that binds them into a dual strand structure called the double helix. These molecules form two sets of base pairs: A-T and G-C. Each base from the base pair is part of an opposing strand (Zvelebil, 2008)

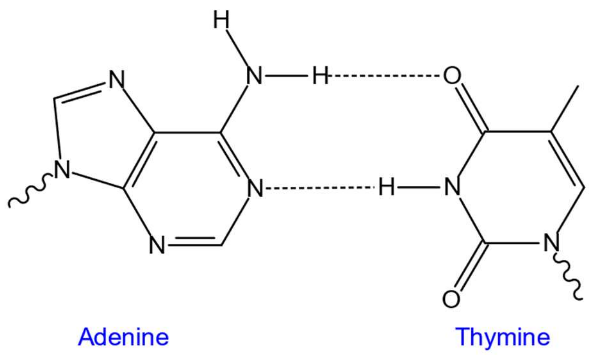


Figure 1 A-T base pair (A-T DNA base pair, 2007)

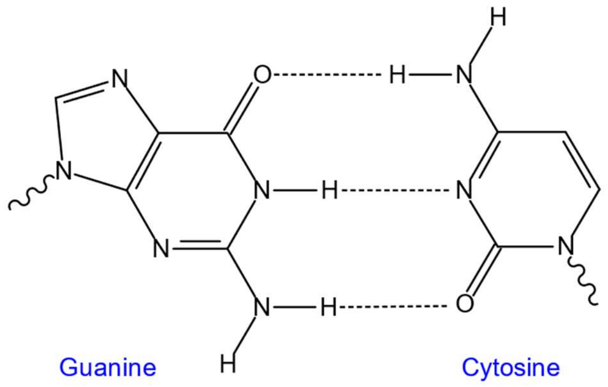


Figure 2 G-C base pair (G-C DNA bae pair, 2010)

While A-T base pair has two hydrogen non-covalent bonds as evident from figure 1, the G-C pair has three non-covalent hydrogen bonds (figure 2). This makes the bond between G and C stronger and therefore harder to break; this fact is important for several reasons into which we will delve later on. At the same time, both types of bonds are still relatively weak as they are non-covalent, therefore much less energy is required to destabilize and break the non-covalent bonds between the two parts of the helix (horizontally) than it is needed to break apart individual strands (vertically).

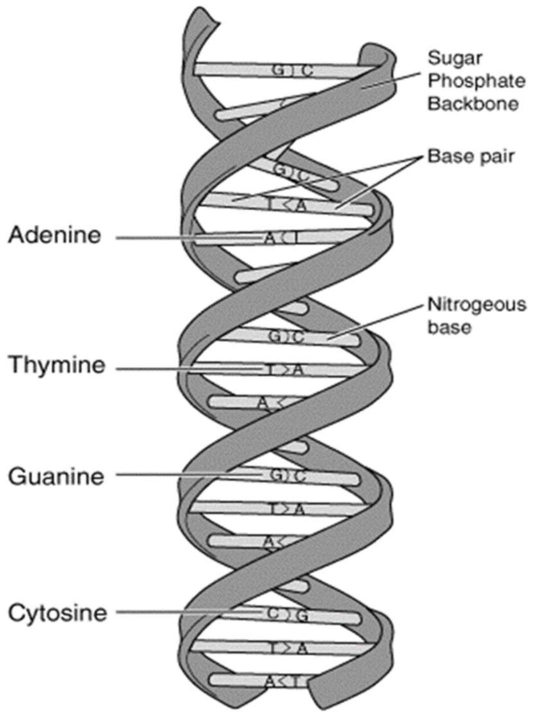


Figure 3 DNA double helix structure (DNA structure detail, b.r.)

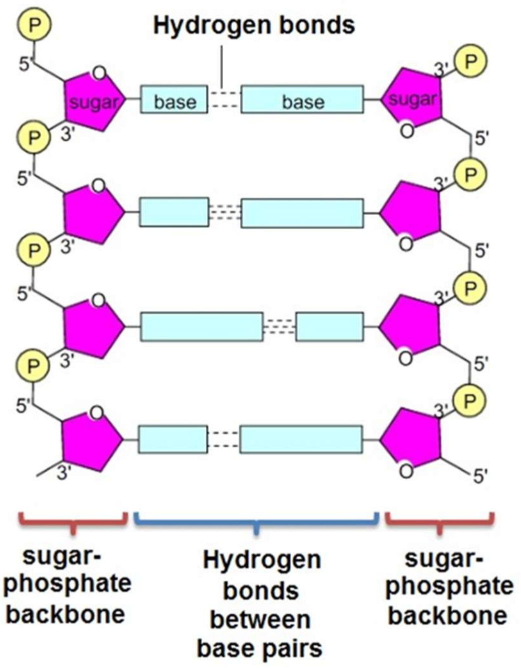


Figure 4 DNA double helix chemical structure (DNA structure detail, b.r.)

This fact is important during transcription and translation, because the enzymes participating in those processes split the strands effectively by “unzipping” the double helix (Zvelebil, 2008).

During the process of transcription, complementary DNA strands are separated by an enzyme called helicase and transcribed by DNA-dependent RNA polymerase into RNA (ribonucleic acid), which is a molecule similar to DNA. RNA differs from DNA in (i) replacing the T base with uracil (U), (ii) replacing deoxyribose with ribose and (iii) being single stranded.

DNA-dependent RNA polymerase reads the so-called anti-coding strand from 3’ to 5’ end to achieve RNA strand in 5’ to 3’ orientation. The coding strand in DNA is defined as the strand corresponding to translated RNA sequence with U replaced with T (Zvelebil, 2008).

The translation process enables the translation of the messenger RNA (mRNA), which is the type of RNA transcribed from a protein coding gene. A particular gene is considered expressed once it is used to synthetize a functional product; this product can ¨í (dividing one DNA molecule into two by splitting its complementary strands and recreating their respective complements) followed by DNA transcription and RNA translation processes along with their inner workings is referred to as “Central dogma of molecular biology“ (Figure 5).

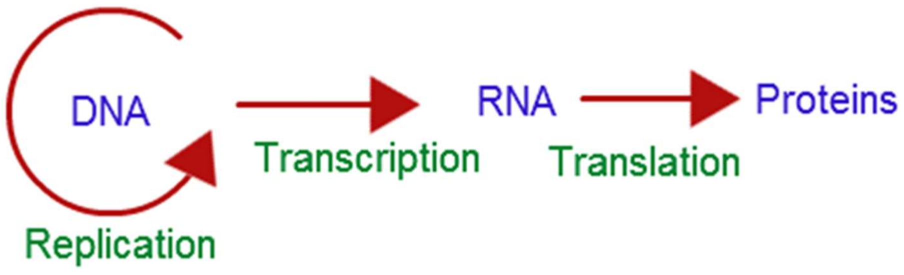


Figure 1 Central dogma of molecular biology (Central dogma of molecular biology, 2013)

2.1.2 DNA sequence

The structure of a DNA (or nucleotide) sequence is defined by an order in which individual bases appear in a DNA strand (either strand can be used since they are complementary) from the 5’ end, which is the end of the strand with a phosphate group on the 5th carbon of the deoxyribose molecule, to the 3’ end, the one with a free hydroxyl group on the 3rd carbon of the deoxyribose molecule (Figure 6) (Zvelebil, 2008).

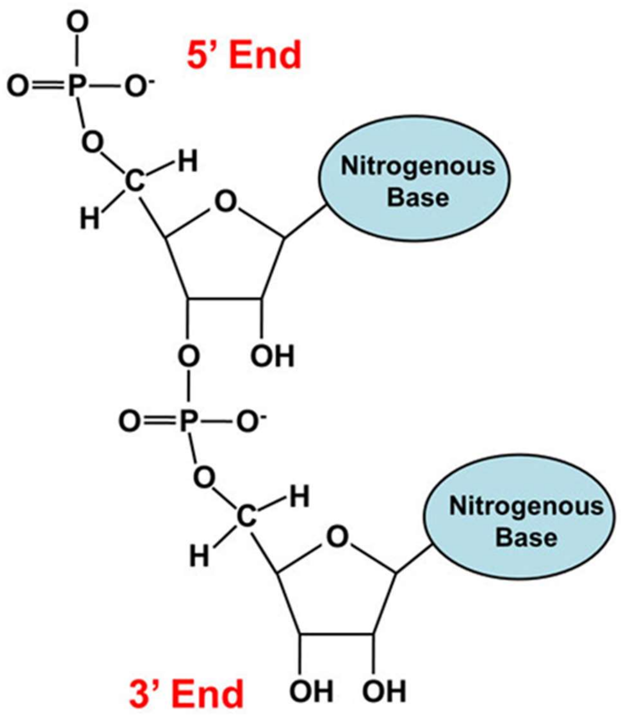


Figure 2 DNA orientation (DNA 3' 5' end, 2013)

This sequence is traditionally represented as a string of characters using G, C, A or T as abbreviations of respective base pairs and other characters for different levels of uncertainty (IUPAC-IUB Comm. on Biochem. Nomencl, 2002).

2.1.3 FASTA format in bioinformatics

In bioinformatics, sequences are most often saved as files in the FASTA format, which contains a sequence string in text format, with a header that provides an identification of the given organism. The header usually consists of the species name and designation of the genome or its part. Furthermore, the multi FASTA format is often used; it combines more than one FASTA records with corresponding headers in one file. This format is typically used with incomplete genomes (which still represent vast majority of genomes in the NCBI database) (FASTA format - NCBI, b.r.). In the following example, you can see a FASTA file with a header containing the accession number for the NCBI nucleotide database along with a species name and strain designation.

```
    >CP014266.1 Acinetobacter baumannii strain Ab421_GEIH-
2010 genome
    TGTGGATAACTTGGGTAGAATGGCGACCCCTTCTCATCAGGAAGGGTTAATCTTT
AAATGATTTGAATTTAAAACGCAGACATAGGGGATACACATGCTTTGGACAGACTGCTT
AACTCGCTTGCGACAAGAGCTCTCTG
    ATAACGTCTTTGCGATGTGGATTCGCCCTTTAGTAGCTGAAGAAGTAGAGGGGAT
ACTACGTCTCTATGC
    TCCTAATCCTTATTGGACGCGTTATATTCAAGAGAATCATTTAGAGTTAATTTCT
ATATTGGCTGAACAA
```

Figure 3 Ungapped sequence in FASTA format

Example in Figure 7 illustrates an ungapped sequence, which is not always the case. Many sequences end up with an incomplete sequence with possible errors, gaps or unclear bases. In a gapped sequence, gaps are substituted with letter “N” (one per one missing base) (Figure 8).

```
    >CP014266.1 Acinetobacter baumannii strain Ab421_GEIH-
2010 genome
    TGTNNNNNACTTGGGTNNNNNNGCGACCCCTNNNNATCAGGAAGGGTTAATCTTN
NAATNNTTNGAATNNAANNNGCAGACATAGGGGNTACACATGCTTTGGACAGACTGCTT
AACTCGCTTGCGACAAGAGCTCTCTG
    ATAACGTCTTTGCGATGTGGATTCGCCCTTTAGTAGCTGAAGAAGTAGAGGGGAT
ACTACGTCTCTATGC
    TCCTAATCCTTATTGGACGCGTTATATTCAAGAGAATCATTTAGAGTTAATTTCT
ATATTGGCTGAACAA
```

Figure 4 Sequence with gaps in FASTA format

The amount of characters inserted should be the same as the amount of undetermined bases or correspond to gap length.As the actual gap length is not always known, it is common practice to use a string of 100 N characters instead (FASTA format - NCBI, b.r.). There are other characters that can be used to specify the level of uncertainty; these are listed in Table 1 (IUPAC-IUB Comm. on Biochem. Nomencl, 2002). Note that the X (standing for A, G, C or T) character is rarely used.

G	Guanine
A	Adenine
T	Thymine
C	Cytosine
R	A or G (Purine)
Y	C or T (Pyrimidine)
M	A or C (Amino)
K	G or T (Ketone)
S	C or G (Strong)
W	A or T (Weak)
H	Not G
B	Not A
V	Not T
D	Not C
N	Any (Undetermined)

Table 1 FASTA allowed characters

In this thesis, DNA sequences used will be presented exclusively in FASTA format, which will also serve as the only allowed input format for the resulting application.

2.2 Comparative genomics

2.2.1 Significance of bioinformatics and computational biology for sequence data processing

Due to tremendous increase in speed, reliability and consistency of DNA (and RNA) sequencing, the increase in available sequence data is astronomical even when looking back only a few years (Heather, 2016). With advent of third generation sequencing, it is possible to obtain whole-genome sequences in matter of hours at a relatively low cost and with low enough error rate (Land, 2015).

The resulting situation sees an exponentially increasing amount of unprocessed sequence data, which on their own have no value, therefore it is necessary to interpret them and understand their significance. This is where computational biology and its tools come in.

Inception of comparative genomics with advent of new technologies such as DNA sequencing, various bioinformatics methods and algorithms for sequence comparison and analysis has led to previously unimaginable advances in speed and efficiency of taxonomic classification. One of the most influenced organism groups was the bacteria domain, which also happens to be quantitatively the largest domain, with estimated number of distinct species between hundred billion and one trillion (Locey, 2016). An estimation that is somewhat supported by new bacteria species and subspecies being discovered on regular basis.

This thesis is focusing on the application of computational biology on bacterial DNA sequences and determining their relatedness by computing the selected genomic parameters, also known as comparative genomics. This is a universal approach not limited to bacteria domain, instead it is applicable to any living organism and even viruses (RNA can be studied and compared in similar fashion). It is important to note that phylogeny and taxonomy by extension is just one of many possible applications of computational biology and bioinformatics.

2.2.2 DNA comparison methods in vitro

DNA-DNA Hybridization

One of the first methods for determining the relatedness of given organisms based on their DNA structure and the only method not based not requiring sequence data knowledge that is still relevant today is DNA-DNA hybridization (DDH).

The process starts with labelling query DNA using a photoreactive or radioactive substance. The query DNA is then mixed with the target DNA and heated up to a temperature that results in breaking the hydrogen bonds between bases and therefore separating both DNA molecules into single strands. This melting temperature is noted as T_m. (Rosselló-Móra, 2011). T_m can be further lowered by adding different chemicals to the mixture such as formamide (Bouvier, 2003). T_m as a measurement of DNA stability directly correlates with %GC content, since there are three hydrogen bonds in G-C base pair as opposed to only two in A-T pair. As a result, more energy is required to break the bonds in GC pairs, therefore substituting AT with GC results in raising T_m for the DNA molecule in question.

Mixture is then cooled back to a temperature that allows strands to re-join. This results in creation of DNA hybrids where a double helix is formed with one strand from query and one strand from target DNA. The relative amount of hybrids created is expressed in percent and is one of the two requirements for determining whether or not query and target DNA represent the same species with a cut-off at 70% (Moore, 1987).

In the next step, the mixture is reheated to point of hybrid strand separation. Difference between T_m of a target sequence and T_m of the hybrids (ΔT_m) is the second parameter to determine relatedness at the species level, with the intraspecies values of ≥5 °C (Moore, 1987).

Disadvantages of this method stem from the need for a lot of wet work with expensive and highly specialized lab equipment. But its main flaws are the inability to exactly replicate conditions in which the experiment has been performed (which can lead to skewed and inaccurate results) and an impossibility to create incremental databases.

2.2.3 Sequence based DNA comparison methods (in silico)

There are two options when it comes to sequence comparison.

First one is to use parameters that are calculated for sequence A and compare those values with the same values calculated for sequence B. An example is %GC.

The second one is to calculate pairwise parameters that are relative and pertain only to a given pair of sequences. An example of this is ANI.

The big advantage of the first case is that all we need to know are parameter values for A and B and we can draw comparison from this information with a simple calculation. In the second case, we need to calculate the parameter for each unique pair. This in turn leads to an exponential increase in required computing resources. For example, when we are trying to determine relatedness for a given dataset, we need to calculate the selected parameter for every unique pair of sequences. Of course, the second method tends to be more accurate, since when drawing conclusion from a single parameter, a great loss of information occurs as the sequence in question has been reduced and simplified to be presented as a single value or a set of values.

The following parameters are the most important ones used in comparative genomics and regarded as the gold standards for taxonomic purposes.

GC percentage

%GC is a percentage measurement of amount of G-C base pairs relative to the total amount of base pairs, that are conclusive in regard to being G or C, or not. This means that the only characters used for this calculation are A, C, T, G, S (representing G or C) and W (r. A or T). All other characters like Ns for gaps and other more specific uncertainty symbols are therefore omitted, since their inclusion would skew results. The resulting formula for calculating %GC in any sequence in proper FASTA format is:

$$\frac{\sum G + \sum C + \sum S}{\sum A + \sum C + \sum T + \sum G + \sum W + \sum S} \times 100$$

In an ungapped nucleotide sequence with all bases conclusively identified (only A, C, T or G), this is an equivalent to:

$$\frac{\sum G + \sum C}{Sequence\ lenght} \times 100$$

While this value has been shown to be very similar in closely related bacteria, it can also be similar in phylogenetically very distant bacteria (Nishida, 2012). For this reason (low information complexity), %GC cannot be used alone as a relatedness defining genomic parameter although it is still used to supplement other, more robust parameters or to serve as a preliminary parameter to decide whether further genome analysis should be performed.

Sequence alignment and BLAST algorithm

Sequence alignment refers to a way of arranging two sequences (DNA, RNA or protein) in a way to overlay and compare regions of similarity. The idea behind sequence alignment is that these aligned regions will help to determine functional and/or evolutionary relationship between sequences, therefore estimating their relatedness and functional similarity.

When it comes to the algorithms used to align two sequences there are two main families of them – algorithms based on dynamic programming and heuristic algorithms.

2.2.3.1.1 Dynamic programming algorithms

Dynamic programming is a method that solves complex problems by breaking them down into in to simpler sub-tasks often utilizing recursion. A proper dynamic programming algorithm is guaranteed to find an optimal (best) solution, which is also the main advantage. Their main disadvantage lies in being compute heavy.

2.2.3.1.1.1 Needleman-Wunsch algorithm

Needleman-Wunsch algorithm is one of the first algorithms that were used for sequence comparison (Needleman, 1970). This is a global alignment algorithm, which means that it aligns two sequences in their entirety – from beginning to the end. As such it is suited for aligning two closely related (highly similar) sequences.

Since Needleman-Wunsch is a dynamic programming algorithm, it breaks down the alignment problem into smaller and simpler issues and then reconstructs the solution. The process itself is best demonstrated on an example:

1. Consider two short nucleotide sequences:
   - I. CGTGAATTCAT; the first sequence with the length(**n**) of 11 bases.
   - II. GACTTAC; the second sequence with the length(**m**) of 7 bases.
2. Construct a matrix with dimensions of (**n**+1)×(**m**+1). One additional field is added to each sequence because we need to consider aligning with a gap at the start. Resulting matrix will look like this:

X	_	C	G	T	G	A	A	T	T	C	A	T
_												
G												
A												
C												
T												
T												
A												
C												

Table 2 Needleman-Wunsch blank

3. Determine a scoring schema – this can be user defined and will assign different scores if:

I. Both nucleotides **match** (ex. A – A). In this case our score will be +1.

II. Nucleotides do not match (ex. A – G). Our scoring system will assign -1 to a **mismatch**.

III. We align a base in sequence A with a gap inserted into sequence B or vice versa. This is called the **gap open penalty** and in our example is set to -1.

Why would we do this? This is supposed to represent and **indel** – a type of genetic mutation that results in deleting or inserting a base from/into the genome. Since the idea behind sequence alignment is comparing equivalent regions, we need to consider this case.

4. Initialize the matrix and fill the first row and column. First cell (1;1) is assigned 0, since this corresponds to starting the alignment with one gap at the start of each sequence, which is the same as starting from the beginning with no gaps. The rest of the scores in the first row and the first column represents the number of gaps inserted. For example, to align the third base of the A sequence with the first base of the B sequence, we need to insert two gaps at the beginning of the second sequence, therefore cell (1;3) has the value of -2.

X	_	C	G	T	G	A	A	T	T	C	A	T
_	0	-1	-2	-3	-4	-5	-6	-7	-8	-9	-10	-11
G	-1											
A	-2											
C	-3											
T	-4											
T	-5											
A	-6											
C	-7											

Table 3 Needleman-Wunsch first row/column

5. Fill the matrix. This is a crucial step, now we will fill the rest of the matrix starting from the top left corner based on values of the three neighbouring cells (left, left diagonal and top). We are looking for the highest assumed score out of the three. These assumed scores are obtained by:

I. Adding the match/mismatch score (further denoted as **S**) to the left diagonal cell value.

II. Adding (it should always be 0 or negative) the gap penalty (further denoted as **W**) to the left cell value.

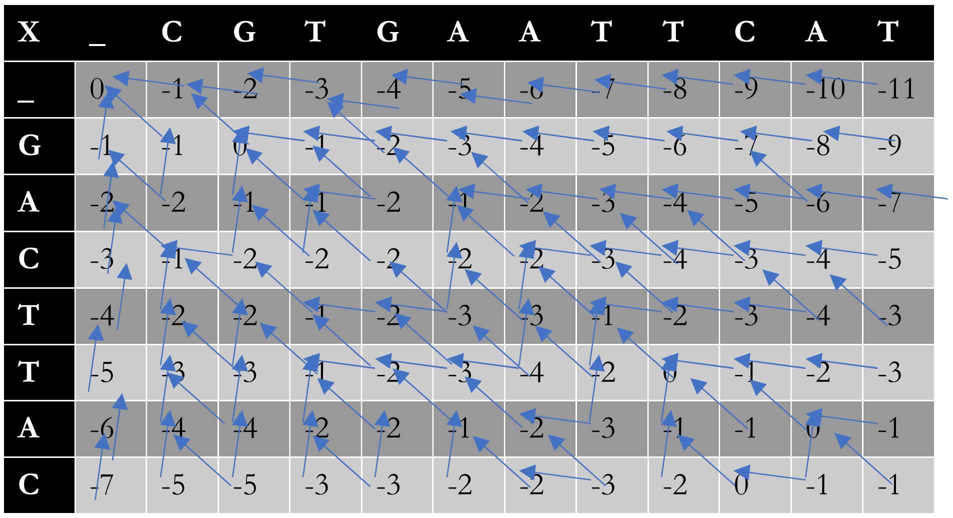
III. Adding the gap penalty to the top cell value.

The mathematical formula for filling each remaining cell is:

$$M_{x,y} = Max(M_{x-1,y-1} + S_{x,y}, \quad M_{x-1,y} + W, \quad M_{x,y-1} + W)$$

where $M_{x,y}$ is the field in question with its respective coordinates.

After determining the maximum value, we will use a pointer to illustrate from which field did the value originated. This information is very important. Quite often, there will be two or even three same (maximal) values. In this case, we will use multiple pointers to all the fields the maximum value was extrapolated from.



X	_	C	G	T	G	A	A	T	T	C	A	T
_	0	-1	-2	-3	-4	-5	-6	-7	-8	-9	-10	-11
G	-1	-1	0	-1	-2	-3	-4	-5	-6	-7	-8	-9
A	-2	-2	-1	-1	-2	-1	-2	-3	-4	-5	-6	-7
C	-3	-1	-2	-2	-2	-2	-2	-3	-4	-3	-4	-5
T	-4	-2	-2	-1	-2	-3	-3	-1	-2	-3	-4	-3
T	-5	-3	-3	-1	-2	-3	-4	-2	0	-1	-2	-3
A	-6	-4	-4	-2	-2	-1	-2	-3	-1	-1	0	-1
C	-7	-5	-5	-3	-3	-2	-2	-3	-2	0	-1	-1

Table 4 Needleman-Wunsch filled

6. <u>Traceback.</u> Now we need to determine the most optimal alignment (or alignments, since there can be more than just one). This step is simple – we follow previously created pointers (arrows) from the right bottom cell of the matrix to the right top cell. Rules for different arrow orientation are following:
   I. Diagonal arrow signifies match or mismatch. This means that in the case of a diagonal arrow, bases are aligned. For example, in our matrix the last bases of both sequences (T and C) are aligned since there is variation in every traceback, because there is only a diagonal arrow originating from the field of their intersection (bottom right).
   II. Horizontal and vertical arrows represent indels. This means that a gap is added to the appropriate sequence. Horizontal arrow adds a gap to the top

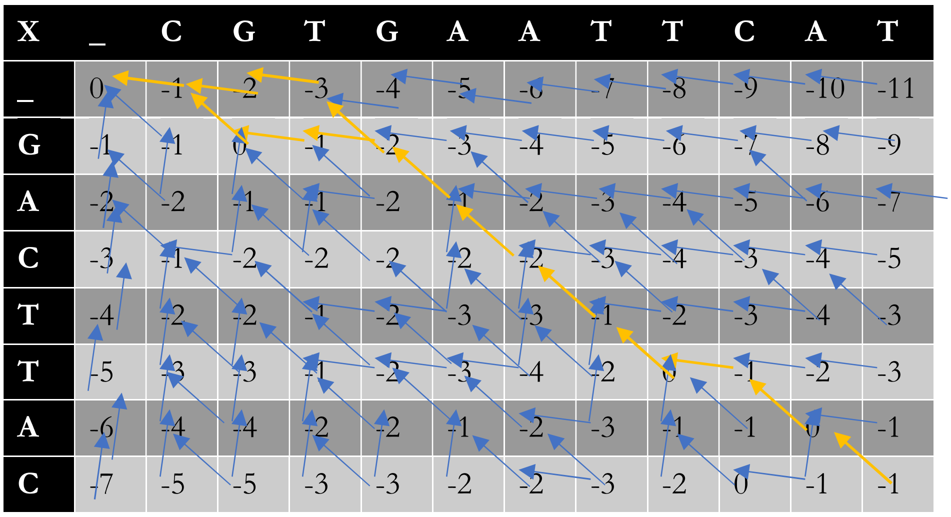
or horizontal sequence, while vertical one adds a gap to the left side or vertical sequence.

III. In the case of multiple arrows originating from a single cell, we are left with two or more alignments. To determine the best alignment or alignments, we will score the traceback route – again the scoring schema here can be user defined.

IV. **Similarity matrices** are a special case of a scoring schema. These allow to specify different scores for alignments of different bases. For example, a G-G pairing can be given a high score of 5, while an A-A pair will only be awarded 2. The two most known and used groups of scoring matrices BLOSUM and PAM (Pearson, 2002) are made for protein sequence alignment. However, such matrices can also be useful when dealing with nucleotide sequences. Reasoning behind this is that some mutations are more likely to occur than others – in DNA the G-T mispair is the most common, since the chemical bonds between bases can rearrange in a way that makes the pairing almost as energy efficient as the "standard" pairs (Kimsey, 2018).

V. **Gap extension penalty** is a special case of gap penalty that is used when there are gaps longer than one base. In case like this, it is not optimal for the penalty to grow linearly, instead gap extension penalty is usually lower than gap open penalty.

Traceback route for our example is denoted with yellow arrows in the following table.



X	_	C	G	T	G	A	A	T	T	C	A	T
_	0	-1	-2	-3	-4	-5	-6	-7	-8	-9	-10	-11
G	-1	-1	0	-1	-2	-3	-4	-5	-6	-7	-8	-9
A	-2	-2	-1	-1	-2	-1	-2	-3	-4	-5	-6	-7
C	-3	-1	-2	-2	-2	-2	-2	-3	-4	-3	-4	-5
T	-4	-2	-2	-1	-2	-3	-3	-1	-2	-3	-4	-3
T	-5	-3	-3	-1	-2	-3	-4	-2	0	-1	-2	-3
A	-6	-4	-4	-2	-2	-1	-2	-3	-1	-1	0	-1
C	-7	-5	-5	-3	-3	-2	-2	-3	-2	0	-1	-1

Table 5 Needleman-Wunsch traceback

7. We can see that there is one cell with two pointers originating from it, thus enabling two branching paths, that represent two equally viable alignments (gap is denoted by X):

   I. C G T G A A T T C A T

X X X G A C T T X A C

II. C G T G A A T T C A T

X G X X A C T T X A C

To determine the most optimal alignment, we will be using a scoring schema of **+5** for **match**, **-1** for **mismatch** and **-2** for **gap open** and **-1** for **gap extension**. This translates to score of **14** for both the first and second alignments. Therefore, both possible alignments are equally viable. (Bioinformatics and molecular evolution, 2005)

2.2.3.1.1.2 Smith-Waterman

The main difference between Needleman-Wunsch and Smith-Waterman algorithms stems from the fact that the former compares two sequences using global alignment, while the latter uses local alignments to compare regions of high enough similarity.

Smith-Waterman algorithm is a dynamic programming algorithm and works in similar fashion to Needleman-Wunsch, any important differences will be accentuated in the following step-by-step breakdown. The same two sequences will be used for comparison:

1. Consider two short nucleotide sequences:
   I. CGTGAATTCAT; the first sequence with the length(**n**) of 11 bases.
   II. GACTTAC; the second sequence with the length(**m**) of 7 bases.

2. Construct a matrix with dimensions of (**n**+1)×(**m**+1). One additional field is added to each sequence because we need to consider aligning with a gap at the start. The resulting matrix will look like this:

X	_	C	G	T	G	A	A	T	T	C	A	T
_												
G												
A												
C												
T												
T												
A												
C												

Table 6 Smith-Waterman blank

3. Initialize the matrix. First row and first column are filled with zeros:

X	_	C	G	T	G	A	A	T	T	C	A	T
_	0	0	0	0	0	0	0	0	0	0	0	0
G	0											
A	0											
C	0											
T	0											
T	0											
A	0											
C	0											

Table 7 Smith-Waterman first row/column

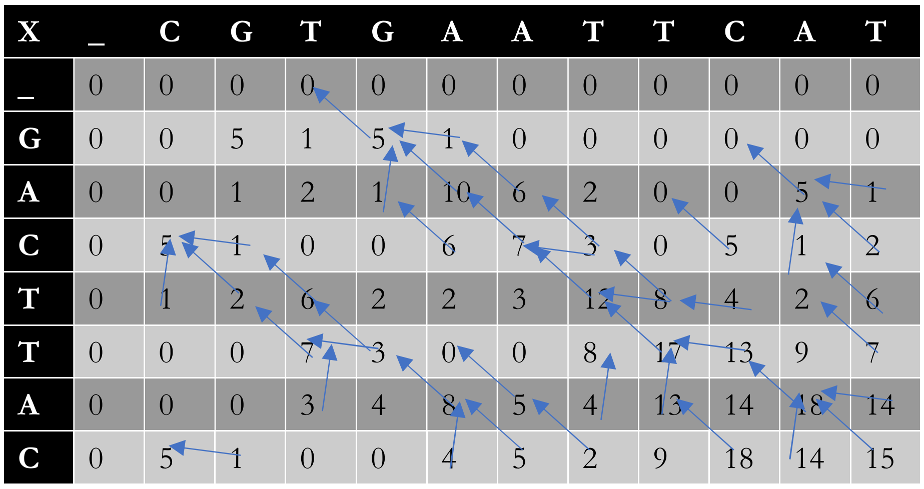
4. Determine scoring schema. Our scoring schema will be **+5** for a match and **-3** for a mismatch. Match/mismatch is again denoted as **S**. For gap penalty, we will use a value of **-4**. Again, this schema can be changed based on different scoring tables or personal preference, however mismatch and gap penalty should always be negative.

5. Fill the rest of the matrix.

   The formula for calculating values of the remaining cells is:

$$M_{x,y} = Max(M_{x-1,y-1} + S_{x,y}, \quad M_{x-1,y} + W, \quad M_{x,y-1} + W, \quad 0)$$

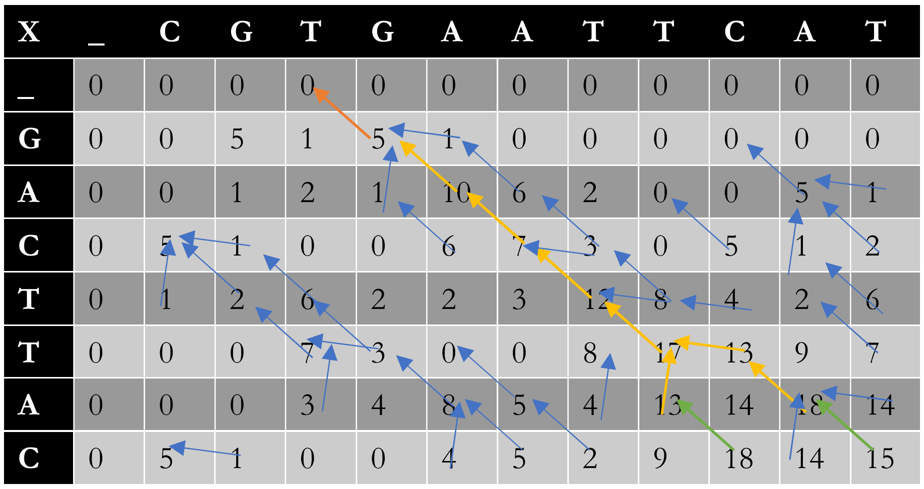
The main difference compared to Needleman-Wunsch lies in adding 0 into the formula. In practice, this ensures that there will be no cells in the matrix that contain a negative value. The aforementioned distinction is what enables the local alignment rather than global, since cells that would otherwise have a negative score signify that there is no similarity between the sequences up to this point. This cell is then set to 0 to ensure that it will have no effect on its successors, which will allow the alignment to start from any position.



X	_	C	G	T	G	A	A	T	T	C	A	T
_	0	0	0	0	0	0	0	0	0	0	0	0
G	0	0	5	1	5	1	0	0	0	0	0	0
A	0	0	1	2	1	10	6	2	0	0	5	1
C	0	5	1	0	0	6	7	3	0	5	1	2
T	0	1	2	6	2	2	3	12	8	4	2	6
T	0	0	0	7	3	0	0	8	17	13	9	7
A	0	0	0	3	4	8	5	4	13	14	18	14
C	0	5	1	0	0	4	5	2	9	18	14	15

Table 8 Smith-Waterman filled

6. Traceback. To start tracing the alignment, we first need to find the cell or cells with the highest score in the matrix. Traceback starts from this cell; if there are multiple occurrences, there will be two or more possible alignments. Alignment ends with a pointer to zero. In the given example, there are two cells with value of 18 which is the highest score in the matrix. Therefore, there will be, at the very least, two possible alignments.



X	_	C	G	T	G	A	A	T	T	C	A	T
_	0	0	0	0	0	0	0	0	0	0	0	0
G	0	0	5	1	5	1	0	0	0	0	0	0
A	0	0	1	2	1	10	6	2	0	0	5	1
C	0	5	1	0	0	6	7	3	0	5	1	2
T	0	1	2	6	2	2	3	12	8	4	2	6
T	0	0	0	7	3	0	0	8	17	13	9	7
A	0	0	0	3	4	8	5	4	13	14	18	14
C	0	5	1	0	0	4	5	2	9	18	14	15

Table 9 Smith-Waterman traceback

(Note that we could start another alignment from the second highest value and another from the third highest etc. The resulting alingments are inferior in their optimality but under certain conditions can provide relevant information)

7. Green arrows represent the alignment start while yellow ones continue the traceback.

   The orange arrow represents the pointer to zero that ends the local alignment. The pointer significance is the same as in the case of Needleman-Wunsch – a diagonal pointer represents aligning of the intersecting bases, while horizontal and vertical ones add a gap to the respective sequence.

As we can clearly see, we are left with two possible alignments. To further determine the most optimal one, we will again score both alignments this time with the same scoring schema that was used to construct the matrix (+5 for a match, -3 for mismatch, -4 gap penalty).

I. G A A T T C A

G A C T T X A

II. G A A T T X C

G A C T T A C

Resulting score is 18 for both alignments, which again signals that both are the most optimal. (Bioinformatics and molecular evolution, 2005)

2.2.3.1.2 BLAST

BLAST or Basic Local Alignment Tool is the most used application for sequence searching and alignment. Its popularity stems from the fact that it uses a heuristic method to approximate, with precise enough results, the Smith-Waterman algorithm all the while making it over 50 times faster than the actual algorithm.

While Smith- Waterman algorithm would be the preferred choice since it guarantees to find the optimal match between two sequences based on the selected scoring matrix (basic matrix "rewards" a nucleotide match with increase the total score by one and penalizes mismatch by decreasing it by one), BLAST due to its heuristic nature does not.

The problem with the Smith-Waterman algorithm lies in its high computational cost. The time and space complexity of the original algorithm is $O(a^2b)$ and $O(ab)$, respectively, where **a** and **b** are the lengths of two sequences used for the alignment (Smith, 1981). Even though the algorithm has been since optimized for time complexity of $O(ab)$ (Altschul, 1986) and space complexity of $O(a)$, where **a** represent length of the shorter sequence (Myers, 1988), it is still not feasible to use with larger datasets.

BLAST algorithm works on a "seed and grow" principle, which means that instead of trying to align one sequence to another as whole, the algorithm looks for short regions that are very similar between the two sequences. In the case of BLAST, the distinction between target and query sequences becomes more important since swapping these roles slightly, but noticeably influences the result, the reason for this will become more apparent in the following algorithm description:

1. <u>(Optional) Remove regions of low complexity</u>

This process (also called dedusting in nucleotide sequences) uses external programs DUST (for DNA) and SEG (for protein) to remove regions of low complexity consisting of long single base repeats or repeated patterns. These regions will make the high complexity ones less significant and therefore can impact the results in a negative way (reduce accuracy) while in most cases, more complex regions are significantly more important for sequence comparison (in DNA, they are much more likely to be functional).

2. Determine initial seed length

   The length of the initial seed is given by the parameter Word length (**k**) – one of the user configurable BLAST parameters. In the context of DNA sequences, a word size of 11 is commonly used, while for protein lower values, starting at three are usually used.

3. Create a list of unique **k** letter words

   The next step is to compile a list of all unique **k** letter words from the query sequence. This is achieved by reading the query sequence starting on the first base and ending on the **k**th base (where **k** is the word length), in the next step reading starts from the second base and this process is repeated until the end of the sequence is reached and the list contains all the unique k letter words that exist within the give sequence.

4. Score the pairs and create HSP list

   After successfully creating the unique **k** letter word list, its members are compared with all the possible **k** letter words (for DNA sequence there is 11^4 of them) and scored according to a scoring matrix like BLOSUM 62 for protein sequences. This is where another parameter Threshold **T** is used. Instead of returning list of all possible pairs and their respective scores, only "High Scoring Pairs" (HSPs) – pairs with score higher than value of **T** – are returned.

   HSPs are in this context **x, y** pairs, where **a** is an existing k-letter word in the query sequence and **y** can be any of the possible k-letter words. Their score is based on their similarity, which is determined by aligning **x** with **y** in an ungapped alignment and summing user defined match/mismatch scores for all **k** bases. Threshold **T** is then applied to the resulting score, if their score is greater or equal **x, y** pair in question is considered a HSP.

   HSP creation is repeated for every unique **k** letter word from the query sequence; the **y** words from all HSPs are then taken and saved into a list as potential seeds.

5. Seeding

   Algorithm then enters the seeding phase searching for exact **k** letter word matches between the potential seed list and the database (target sequence). These matches between query sequence and database are designated as seeds.

6. Seed extension

   After seeding is concluded, BLAST algorithm starts to extend the initially aligned seeds and depending on set gap penalty, mismatch penalty and rewards for HSP matches or connecting with another seed the alignment score increases or decreases. The extension will stop when the cumulative score for the alignment in question drops below **T** level.

(Mclean, 2004) (Altschul, 1990)

Resulting alignments can be formatted to one of many BLAST applications output formats and further analysed and interpreted (BLAST® Command Line Applications User Manual, 2008).

Average Nucleotide Identity based on BLAST (ANIb)

Average nucleotide identity is a genomic parameter created to simulate DNA-DNA hybridization in silico (Richter, 2009). The implementation in this thesis is based on the BLAST algorithm described in the previous section.

The implementation of ANI based on BLAST requires some extra steps; the exact methodology was described by Klappenbach et al. as follows:

"The genomic sequence from one of the genomes in a pair ('the query') was cut into consecutive 1020 nt fragments. The 1020 nt cut-off was used to correspond with the fragmentation of the genomic DNA to approximately 1 kb fragments during the DDH experiments. The use of different cut-offs (e.g. smaller fragments) did not notably modify our results (data not shown). The 1020 nt fragments were then used to search against the whole genomic sequence of the other genome in the pair ('the reference') by using the BLASTN algorithm the best BLASTN match was saved for further analysis. The BLASTN algorithm was run using the following settings: X=150 (where X is the drop-off value for gapped alignment), q=−1 (where q is the penalty for nucleotide mismatch) and F=F (where F is the filter for repeated sequences); the rest of the parameters were used at the default settings. These settings give better sensitivity than the default settings when more distantly related genomes are being compared, as the latter target sequences that are more similar to each other.

....

The ANI between the query genome and the reference genome was calculated as the mean identity of all BLASTN *matches that showed more than 30% overall sequence dentity (recalculated to an identity along the entire sequence) over an alignable region of at least 70% of their length. This cut-off is above the 'twilight zone' of similarity searches in which an inference of homology is error prone because of low levels of similarity between aligned sequences. Therefore, we can assume that only homologous DNA fragments were considered in our calculations."* (Klappenbach, 2007)

Based on the cited methodology the step-by-step process for computing the ANIb values in a pairwise comparison is:

1. Load the two sequences **A** as the query and **B** as the target or database.
2. Chop the **A** (query) sequence into 1020 base fragments (the last fragment will likely end up shorter).
3. BLAST every fragment of **A** one by one (ideály concurrently using multithreading) against the complete **B** (target) sequence using the parameters specified above.
4. Check the results:
    I. If the length of the resulting alignment is at least 70% of the overall fragment length (fragment length divided by the length of the alignment is at least 0.7).
    II. If the % nucleotide identity recalculated to the length of the whole fragment is above 30% (number of matched nucleotides divided by the fragment length is above 0.3). (
    Note that the selected approach is one of the two possible interpretations. The other would be dividing the length of the alingment by the fragment length. This would be a slightly less strčit in case of a gapped alignment (very common). Ultimately the authors educated opinion is that the 30% cut off is very low as is and therefore is better to use the stricter method.

5. Every fragment that fulfils these two conditions will be used to calculate the ANIb value by calculating the total mean of their identity percentages.
6. Repeat all of above only with the roles for **A** and **B** reversed.
7. Calculate the mean of the two resulting values to get the bi-directional ANIb value for the examined sequence pair.

ANIb score corresponding to a DDH value of 70% and therefore to the boundary for determining whether are the two examined prokaryotic organisms part of the same species, has been set (based on extensive experimentation) between 94-96%. It has been proven further that ANI score tightly correlates with DDH results for phylogenetic purposes and therefore is a full-fledged alternative to DDH, while superseding it by eliminating almost all of its drawbacks such as high cost, time required to perform the experiment, man-hour demand and other uncontrollable factors that can potentially influence results (Richter, 2009).

It is worth noting that another option for calculating ANI score is available based on the MUMmer algorithm, that boasts a speed advantage over ANIb with only a very slight decrease in accuracy when comparing organisms that appear to diverge at the species level, (≈ANI score of 90% and lower).

ANIb is presently regarded as the new "gold standard" for prokaryotic species definition and it is quickly becoming the most used parameter for taxonomic and phylogenetic studies (Rosselló-Móra, 2015).

Tetranucleotide (oligonucleotide) frequency correlation]

Oligonucleotide frequency in each sequence is determined by the number of times a unique oligonucleotide appears in the sequence. Unique oligonucleotide is defined as a distinct nucleotide sequence of length **k,** where **k**=1 for mono nucleotide (single A, C, T or G base), **k**=2 for dinucleotide (pairs like GC, TG, GT, AT etc.), **k**=3 for trinucleotide (GCT, TGC, AAA, CGC…), **k**=4 for tetranucleotide (AGACT, TTGA,CGCT, AAAG..) and **k**=n for n-nucleotide.

While the fact that oligonucleotides carry species-specific signal was experimentally proven, suggesting that different species are biased towards under and over representing different oligonucleotides and this profile is indeed species specific, the reasoning behind this is still unknown. It has also been demonstrated that longer oligonucleotides carry more of this signal than the shorter ones. Calculation of the tetranucleotide frequencies and their correlation within examined dataset seems to provide a good balance between performance cost and strength/reliability of the phylogenetic signal. (Richter, 2009).

Tetranucleotide correlation coefficient between two DNA sequences is calculated using method published by Teeling et al (Teeling, 2004), which itself is derived from more universal method for calculating expected oligonucleotide frequency via Markov models published by Schbath et al. (SCHBATH, 1995).

First step was to split the examined sequences to 40 kilobase (4000 bases) long fragments designated as fosmids.

"In brief, all fragments were extended with their reverse complements. The observed frequencies of all 256 possible tetranucleotides and their corresponding expected frequencies were computed for these sequences. The differences between observed and expected values were transformed into z-scores for each tetranucleotide. The similarity between two fosmids was assessed by calculating the Pearson correlation coefficient for their 256 tetranucleotide-derived z-scores." (Teeling, 2004)

Expected tetranucleotide frequencies are computed by the maximal order Markov model from dinucleotide and trinucleotide frequencies using the following formulas:

$$E(n1n2n3n4) = \frac{O(n1n2n3) \times O(n2n3n4)}{O(n2n3)}$$

$$Z(n1n2n3n4) = \frac{O(n1n2n3n4) - E(n1n2n3n4)}{\sqrt{var(O(n1n2n3n4))}}$$

$$var\big(O(n1n2n3n4)\big) =$$
$$= E(n1n2n3n4) \times \frac{[O(n2n3) - O(n1n2n3)] \times [O(n2n3) - O(n2n3n4)]}{O(n2n3)^2}$$

Where **E** signifies expected frequency of the given n-nucleotide, **O** stands for observed frequency of the given n-nucleotide, **var** is the variance and **Z** is the z-score used to represent the divergence between the expected and observed values. The n1 to n4 stand for the particular bases in the tetranucleotide in question.

Determining whether two fosmids exhibit similar nucleotide over and under representation patterns is realized via calculating the Pearson correlation coefficient for the corresponding z-scores using this formula:

$$p_{x,y} = \frac{\sum XY - \sum X \times \sum Y}{\sqrt{\sum X^2 - (\sum X)^2} \times \sqrt{\sum Y^2 - (\sum Y)^2}}$$

Where the X values are the z-scores of the 256 possible tetranucleotides computed for the first fosmid and the Y values are the z-scores for the second one.

Resulting Pearson's coefficient has been shown to greatly exceed simple GC content calculation when it comes to determining species relatedness (Teeling, 2004). Its values of >0.99 can quite reliably indicate ANI score of 95-96% or higher and by extension species circumscription, however a considerable number of outliers has been reported, where tetranucleotide correlation values suggest both significantly lower and higher relatedness than ANI scores. Therefore, we can conclude that tetranucleotide correlation score, while providing much higher accuracy for phylogenetic analysis than simple %GC calculation, is still not as reliable as ANI score for the same purpose. Nevertheless, since its performance cost is significantly lower than that of ANIb it can be used for preliminary analysis of large datasets and due to its decent accuracy even as an accompanying/control parameter for ANI-based phylogeny (Richter, 2009).

It is important to note that for the purposes of this program the TETRA value will be counted without the artificial fosmid division. This is because, our application differs from Teeling et al. in the fact that we are not interested in the intra-genome values, only inter-genome ones. Furthermore, we will often end up using incomplete genomes, unlike the exclusively whole genome sequences used in the referenced study. For these reasons the division to 40kb fragments is not beneficial or useful for our purposes.

All of the above translates to the following step-by-step process used to determine the pairwise TETRA values:

1. Load the two sequences – **A** and **B** and extend both by their complementary strains.
2. Calculate the z-scores using the previously detailed formula for all 256 tetranucleotides in each sequence.
3. Calculate the Pearson correlation coefficient between the z-scores of **A** and **B**.

2.2.4 Need for an offline and user-friendly application

In previous paragraphs, we went through the most used (presently and in the past) genomic parameters for phylogenetic analysis and classification of bacteria or even prokaryotic organisms as a whole.

With this in mind, it is almost baffling that (to the best of the authors knowledge) there is no offline solution that would facilitate the computation of these parameters while implementing a user-friendly (or even any) graphical interface. The only exception is the JSpecies (Jspecies, 2009) application, that implements a very polished graphic UI. However, this software is ultimately let down by being outdated which translates into not working properly with new versions of NCBI BLAST and suffering from two major flaws: – an incorrect implementation of %GC content calculation and the (probably being an unfixed bug) inability to process datasets bigger than 20 sequences (FASTA files), when comparing all possible pairs.

While there is a more than fair selection of command line applications and freely available scripts that can be used, this requires basic or even advanced computer skills, which still is not a given for many biologists. Another possible solution is again a rather extensive selection of web-based applications, that while solving the problem of missing graphical UI and ease of use come with a different issue – a dataset size limit typically falling between 10-20 sequences per project. In the case of the improved and reworked JSpeciesWS (JspeciesWS, 2015) web application, this limit is 15 sequences per project, which is inconvenient for processing large sets of data. Another possible drawback is that when the server in question is experiencing intense traffic, overload might occur leading to much longer computation time than expected.

All of the above-mentioned leads to a clear conclusion – there are no solutions that would satisfy all of the requirements.

Even if we consider only the three most important requirements:

1. ANIb calculation for unlimited datasets.
2. Offline functionality.
3. Usability for regular computer users.

The list of applications that would satisfy all three boxes requirements is still empty.

.

3 Goals

The goal of this thesis is to design, implement, test, and release a bioinformatics toolkit, the main functionality of which lies in comparing genomic parameters between the nucleotide sequences of bacterial whole genomes.

This application is intended as a taxonomic tool, helping with classification and taxonomy of bacteria. It can be used to compare large sets of genome sequences as well as their selected subsets, all the while being simple to use and navigate which should result in its practical usability even for inexperienced lab and research personnel.

Partial goals derived from the primary one are:

- To review theory and research behind selected in silico sequence-based genomic parameters, their predecessors and alternatives as well as their in vitro counterparts and predecessors to gauge their usability, advantages and limitations.
- To gain deep understanding of mentioned algorithms and their inner workings.
- Based on this review, to decide which parameters should be prioritised and which (if any) should be omitted.
- To analyse the need for configurability to provide only relevant options while avoiding unnecessary options.
- To design a project-based saving system that will ensure data preservation and accessibility while enabling the program to recover from non-standard termination without data corruption and loss of already computed results.
- To design application structure while taking speed and efficiency into consideration, ideally using multiple processing threads.
- To design the application architecture that will fit outlined goals while being compatible with personal computers based on x86 processor architecture and running the Microsoft Windows operating system.
- To implement a solution based on the outlined design and its functional and non-functional specifications.
- To document the implementation in a reasonable degree of detail (i.e. make sure that the source code snippets are not the dominant part of this thesis's practical part).
- To create a simple user manual that explains the application functionality from a practical standpoint.

4 Application design

4.1 Development process

The *"Waterfall Software Development Cycle"* was used as the application development model. This cycle consists of following phases:

1. Requirement analysis.
2. Design.
3. Development.
4. Testing.
5. Maintenance.

For the purpose of this thesis and the application in question only first 4 phases will be considered as the maintenance phase will take place after the publication and throughout the application life cycle.

4.2 Requirement Analysis

4.2.1 Requirement specification

Requirements have been based on the feedback of researchers and lab workers from the Laboratory of Bacterial Genetics (LBG) at the Czech National Health institute (SZU) who will serve as testers and first users as well as consulted with the thesis advisor.

Requirements have been split into two categories – functional and non-functional.

In the following two lists of specifications, only the main and most important requirements are outlined. Even though there are many more less significant requirements, listing all of them would serve as clutter making this thesis unnecessary long and hampering readers orientation.

More notable challenges and requirements that made themselves apparent only during the implementation process will be discussed in the corresponding chapter.

4.2.2 Functional specification

This section details the functional requirements of the application. In other words, this is what the application will actually be "doing" (i.e. tasks that the program will perform). This does not include architectural, environment or hardware specifications.

- Resulting system will come in the form of a single application that will handle all of the programs functionality.
- The application will enable any user to:
    - Set a workspace folder for future projects.
    - Create a new or open an existing project.
    - Delete a project.
    - Import any number of sequence files in .fas, .fasta, .fsa or .txt format.
    - Manage this file collection with the option to add and remove files.
    - Set specific options for BLAST algorithms to influence ANIb calculation.
    - Set a path to the blastn executable file that needs to be present on the host machine in order to calculate ANI based on the NCBI implementation of the BLAST algorithm.
    - Compute selected genomic parameters, depending on their nature either for the single sequence (%GC) or as a pairwise comparison (ANIb, TETRA).
    - Reset (delete) computed results within the project.
    - Export results in the form of a properly formatted .csv file.
    - Exit the program.
- The application should be able to manage its folder and file structure in an efficient way – i.e. not leaving any residual folders and files after the project is deleted and erasing all the temporary files (this pertains mostly to chopped query fragments during ANIb calculation) when the calculation is finished.
- The program should be able to cope with non-standard termination well, not losing any results except for the last ANIb calculation that was in fragmented state during termination – in this case temporary files will be deleted when the calculation is initiated again.
- The program needs to be able to process virtually unlimited number of files with the total number of pairwise comparisons realistically going into thousands.
- If possible, the application should take advantage of multicore systems to increase its speed.
- Visual interface should be simple and clearly labelled to make user orientation easy.

The above-mentioned requirements are already application specific, however more refinement is needed in several cases and some compromises will have to be made, especially when it comes to multithreaded nature of the program.

When it comes to priorities the most important one is the ANIb calculation. Since this is a parameter, that has become a golden standard in bacterial taxonomy it needs to work reliably and consistently.

4.2.3 Technical specification

This section specifies things such as the environment in which the application should run, hardware it should run on and software needed to run it. Technical or non-functional specification is used to define the application properties rather than its functions.

- **Programming language** of choice is **C#** using **Microsoft .Net Framework**, with **Microsoft Visual Studio** as the integrated development environment (**IDE**).
- Application will be developed as a Windows Presentation Foundation (**WPF**) project and will use its respective libraries. While deciding between using WPF and Windows Forms library, pros and cons of both were considered and choice to use WPF was made due to its more modern feature set and higher flexibility.
- The application should run under Microsoft Windows systems. While compatibility with Windows Vista should theoretically not be a problem it is not guaranteed, and it will not be tested. Compatibility is guaranteed with Windows 7 from Service Pack 1 onwards and all versions of Windows 10.
- This application does not require an internet connection to run.
- In order to run the program Microsoft .NET Framework should be installed in its latest version. This is however fixed by the system itself if needed.
- For the calculation of the ANIb genomic parameter, NCBI BLAST - preferably in the latest version is required, more precisely having the blastn executable file is sufficient.
- The application language is English.

4.2.4 Use cases

While the scope of this application is very focused and narrow, we will still go over a few model use cases. This should enable further validation and/or reassessment of the application functional specification.

Actors

Identifying actors (entities that interact with the program and use its functionality) is rather simple since there is only one actor – any user.

There are no special privileges or user groups with varying degrees of control, so any user has access to all of the applications functionality.

Use cases – project management

Actions used to create and manage a project perpetuated by the user include project creation, opening an existing project, deleting a project, importing sequence files into the project, adding files into the project, removing files from the project and changing project specific and global settings (Figure 9).

- **Create a project** – User has an option to create and give a custom name to a new project.
- **Open an existing project** – User has an option to open an existing previously created project.
- **Delete a project** – User can delete any existing project by first opening it and then selecting the delete option.
- **Import sequence files** – After opening or creating a project, user can populate the project with sequence files whose names will be displayed afterwards.
- **Add sequence files** – User can add sequence files to the project with the UI updating accordingly.
- **Remove sequence files from the project** – User can remove sequence files from the project with the UI updating accordingly.
- **Change settings** – User has an option to change global settings variables such as workspace folder and path to the blastn executable file.

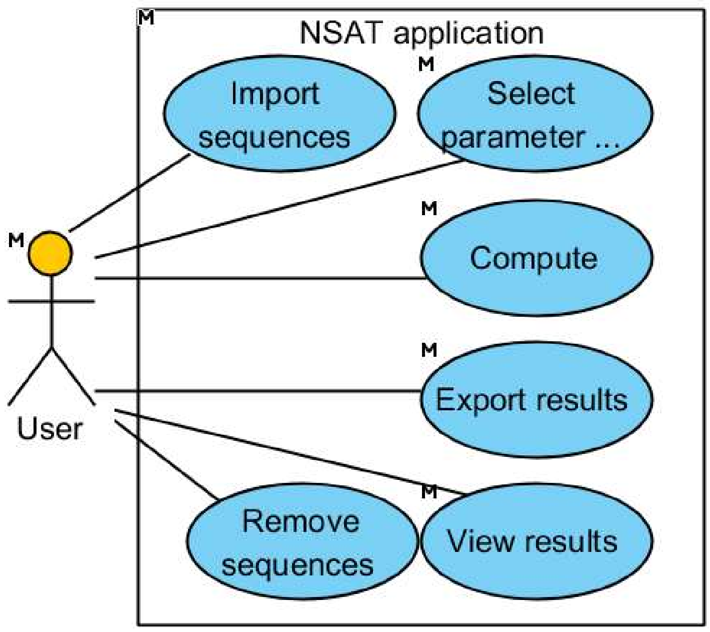


Figure 5 Uc Project management

Use cases – calculation

User actions performed in order to calculate genomic parameters for a given set of sequence files including selecting the parameter, selecting the pairs or individual files from the project to include in the computation itself, computing the parameters, exporting and viewing the results.

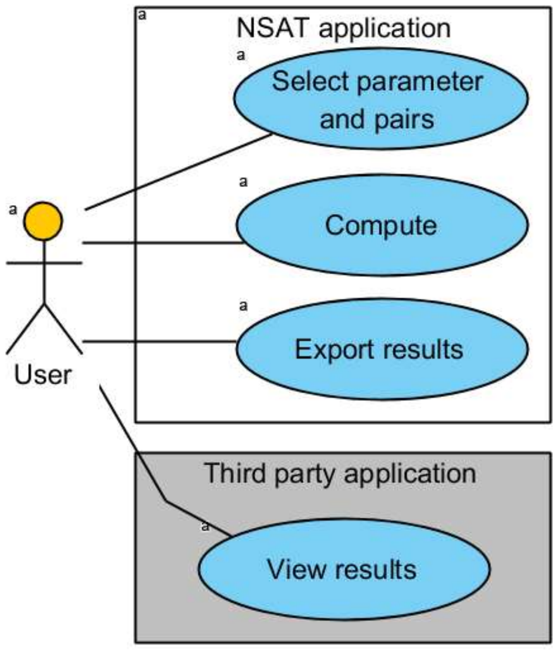


Figure 6 Uc Calculation

- **Select parameters and pairs to compute** – User can select the parameter he wants to calculate and pairs of sequences to be included in this calculation from the list of all available sequences for pairwise comparisons or %GC calculation.
- **Compute** – User then has an option to either confirm or cancel the selection starting or cancelling the computation process.
- **Export results** – User has an option to export the results for the finished calculations to a location of his choosing in .csv format.
- **Viewing the results** – User is free to select a software to view the exported files based on his/her own preference since the output file format is common and widely supported.

4.3 Design

4.3.1 Domain model

This model (Figure 11) illustrates the main project entity and its relations with other objects from a domain perspective. It was modelled using the Universal Modelling Language (UML) via the Visual Paradigm software.

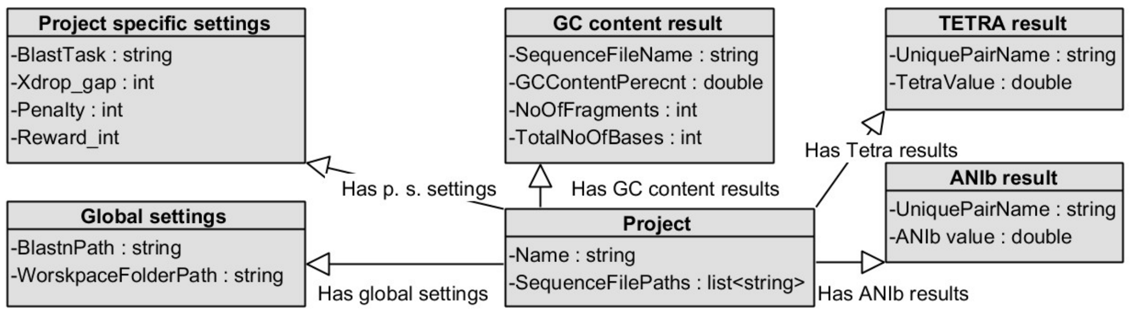


Figure 7 Domain model

4.3.2 Architecture

In terms of application architecture, the program design makes use of three layers:

1. **Presentation layer** – Handles graphical user interface, user interaction, form validation etc. Interfaces with the application layer.
2. **Domain or application layer** – Handles the application logic such as different algorithms and data processing. Interfaces with both the data layer and the presentation layer.
3. **Data layer** – Stores application data. Interfaces with the application layer.

In short, the presentation layer enables the user to make request, this results in calling a function or method in the application layer which then accesses the necessary data using the data layer and if applicable passes the result back to the presentation layer resulting in a visual change.

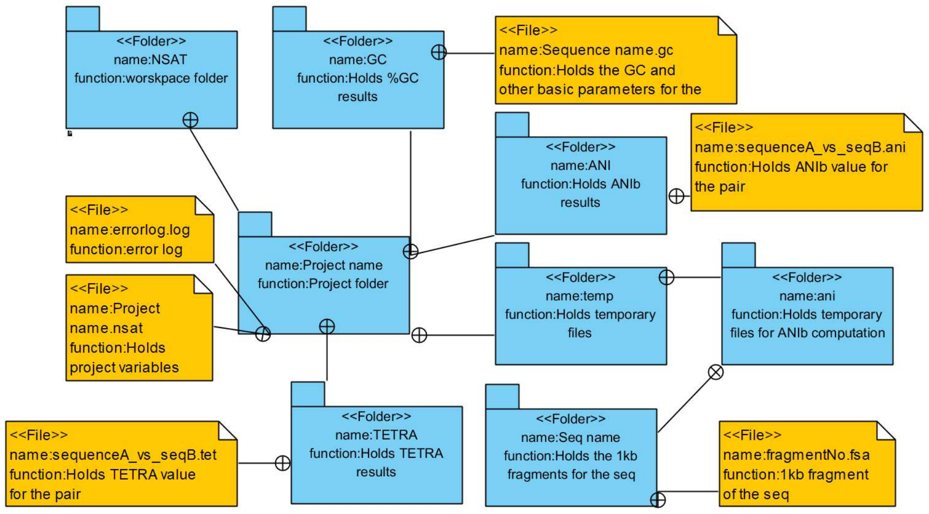
An argument could be made that a domain layer should be differentiated and while the distinction could be made perhaps in regard to the application layer being the one including algorithms and methods, while the domain layer should govern the usage of data layer methods for populating objects and variables. However, when it comes to this project the author decided against making this distinction, since trying to distinguish the two would only lead to confusion and would not serve any real purpose.

In the case of this application, no database in the traditional sense will be used; instead the data are going to be stored in and loaded from a directory/file structure using custom file types for configuration and project data and computational results**.** This decision was made based on the fact that a separate database service is neither desirable nor required and would likely introduce unnecessary performance issues.

Data layer

The data layer defines and creates the file structure and handle accessing and saving the required data.

The UML diagram (Figure 12) shows the file structure end respective file content that will be used in place of a traditional database.



<<Folder>>
name:NSAT
function:worskpace folder
<<Folder>>
name:GC
function:Holds %GC results
<<File>>
name:Sequence name.gc
function:Holds the GC and other basic parameters for the
<<Folder>>
name:ANI
function:Holds ANIb results
<<File>>
name:sequenceA_vs_seqB.ani
function:Holds ANIb value for the pair
<<File>>
name:errorlog.log
function:error log
<<Folder>>
name:Project name
function:Project folder
<<File>>
name:Project name.nsat
function:Holds project variables
<<Folder>>
name:temp
function:Holds temporary files
<<Folder>>
name:ani
function:Holds temporary files for ANIb computation
<<File>>
name:sequenceA_vs_seqB.tet
function:Holds TETRA value for the pair
<<Folder>>
name:TETRA
function:Holds TETRA results
<<Folder>>
name:Seq name
function:Holds the 1kb fragments for the seq
<<File>>
name:fragmentNo.fsa
function:1kb fragment of the seq

Figure 8 Data layer

Application layer

This layer includes all of the algorithms and associated methods for computing the genomic parameters as well as project management.

The following UML diagram depicts the example ANIb class.

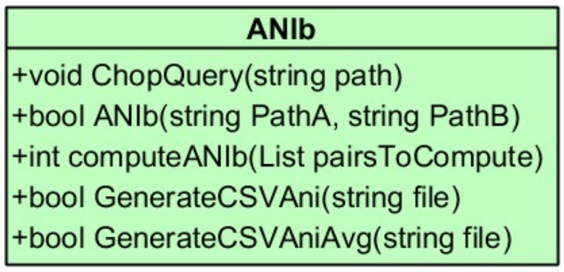

ANIb
+void ChopQuery(string path)
+bool ANIb(string PathA, string PathB)
+int computeANIb(List pairsToCompute)
+bool GenerateCSVAni(string file)
+bool GenerateCSVAniAvg(string file)

Figure 9 ANIb class

Presentation layer

The presentation layer will use WPF controls implemented automatically by the IDE in the xml language and databinding to construct the user interface. Function ‹calls› and ‹variable changes› initiated from the UI will be handled via different Events (for ex.: OnTextChange, OnClick, OnSelectionChange). Events in this context could be described as a code block that is executed on the program detecting an interaction with the user interfaces such as clicking a button, clicking a menu item, writing text into an enabled text box or checking a check box or toggling a radio button.

4.3.2.1.1 UI design

The interface should be simple, concise and focused, enabling only methods and settings that are relevant for the application functionality while keeping the visual flare to a minimum.

After the application starts, the main interface window will be initialized with the following layout:

At the top of the window, there will be a horizontal menu containing six items with dropdown submenus. These will be:

I. **File** – Active at all times. Menu item enabling project control with options to:
    a. **New project** – Active at all times. Create a new project. Opens a confirmation dialog with the possibility to choose the project name.
    b. **Open project** – Active at all times. Open an existing project via standard Windows Open File Dialog.

   c. **Delete project** – Active only when a project is loaded. Delete the currently loaded project. Opens a confirmation dialog.
   d. **Exit** – Terminate the program. Opens a confirmation dialog.

II. **Import** – Active when a project is loaded. Import sequences:
   a. **From local files** – Import a sequence file in any of the supported formats from a local storage device. Realized via Windows Open File Dialog with multiselect.

III. **Compute** – Active only when a project containing at least one sequence is loaded. Menu item consisting of two submenus:
   a. **ANIb/TETRA** – Active only when there are two or more sequence imported into the loaded project. Allows the user to compute ANIb and TETRA genomic parameters. Opens a list of all possible pairs with checkboxes for both parameters and confirmation/cancel buttons. Further options such as select all, select none and invert selection are present. The two parameters are grouped together for convenience since computing both for a given dataset is a common practice.
   b. **GC content** – Compute the %GC for selected sequences. Opens a list with all the imported sequences and check boxes for each, options to confirm/cancel are present. Same selection options as above are supported.

IV. **Results** – Active only when a project is loaded and results for some or all of the imported files are available. Menu item consisting of three submenus:
   a. **ANIb** – Active only when ANIb results for one or more pairs are available. Upon expanding three more options are available:
      i. **Full matrix** – Export a full result matrix in the .csv format to a local storage location specified by the user for the ANIb algorithm with values for both alignment orientations (a as query/target, b as target/query) if available.
      ii. **Averaged matrix** – Active only when both alignment orientations are computed for all of the present results. Export a matrix in the .csv format to a local storage location specified by the user with both alignment orientations averaged to a single value – a standard output for practical use of an ANI based comparison.
   b. **TETRA** – Active only when TETRA results for one or more pairs are available. Upon expanding two more options are available:
      i. **Matrix** – Export a matrix in a .csv format to a local storage location specified by the user containing tetranucleotide

frequency correlation coefficients between the computed sequence pairs.

c. **GC content** – Active only when one or more results for %GC are available. Export a matrix in .csv format to a local storage location specified by the user containing %GC value, sequence length and number of fragments (this will be >1 in case of multi FASTA files).

V. **Tools** – Active at all times. Expands into two more items:

a. **Settings** – Opens a new window allowing the user to set (note that the last four settings will be active only when a project is loaded since they are project specific):

i. **Workspace folder** either by inputting the path manually into a text box control or by using a standard Windows dialogue.

ii. **Blastn executable location folder** either by inputting the path manually into a text box control or by using a standard Windows dialogue.

iii. **Blastn algorithm type** by a set of mutually exclusive radio buttons.

iv. **Mismatch penalty.**

v. **Match reward.**

vi. **Xdrop gap final parameter.**

b. **About**

VI. **Reset values** – Active only when a there are is at least one result computed for any of the parameters within the current project. Contains three sub items that represent the corresponding parameters and are active only if the appropriate results are available. This will delete all the results for a selected genomic parameter in the given project. Confirmation prompt will appear.

In the centre – filling the majority of the window real estate will be the **file list**, this will contain the names of the loaded FASTA, .fas or .fsa files. On the right side of the list area a **delete button** will be present, allowing the user to remove any highlighted items from the project.

On the bottom there will be a **status strip** displaying messages about performed tasks and computational progress.

4.3.3 Class structure

Classes in the project are organized in regard to the individual genomic parameters – this spans three separate classes (ANIb, TETRA and GC).

The next separate class is the Project class, which facilitates project management with relevant methods while housing global project variables. It also governs the file structure and enables data access via its global variables for other classes and methods. Finally, it houses the FASTA validation function.

To handle the settings saved in the registry and their default value, Global Settings class is implemented.

Serving as main is the automatically generated Main Windows class. This serves as an entry point for the application calling the needed methods on initialization. It also holds all of the events triggered by interaction with the UI that are implemented mainly in its MainWindow.xaml.cs counterpart.

Additionally, a few other classes are implemented for each of the additional windows such as Settings Window, About window and pair/sequence selection windows for computation.

While the Window classes serve mostly as the presentation layer, the distinction between the application and data layer is less clear and there are overlaps between both.

Rules of the class interaction can be summarized like this: The Main Window class and/or its sub-windows access all other classes – Project class, parameter classes, Global Settings and UPGMA class. Parameter classes and the UPGMA class use Project variables, structures and methods. The Project class accesses only the global settings class.

Overall, the only reason this class structure has been chosen is that the author considered it the most natural and organic way for him to design and implement the application.

5 Implementation

The basic idea for the implementation is to build the fundamental blocks first.

In the first step, three class files were created – ANI.cs, GCC.cs and TETRA.cs. These contain the elements outlined above.

This led to the next step. It was necessary to create project logic in Project.cs in order to handle file structure, collections, saving and loading of project data and computation results. Furthermore, a validation method had to be created in order to determine whether the input files are in the correct FASTA format, count the fragments and prepare the input string for the computation of the respective genomic parameter.

Next, the short Global Settings class was implemented.

The graphic interface in xml using WPF libraries and control logic was created. This includes MainWindow and several other windows with various control elements for project management, settings, computation and exporting results. This is the connecting fabric for all the created modules.

Helping the author to learn the skill needed for this massive undertaking were several books, hours of internet searching and going through different bioinformatics and programming forums and obscene amounts of caffeine. (Compeau, 2015) (Nagel, 2009)

In the following paragraphs, the implementation is briefly described on a class-by-class basis, while problematic and/or interesting parts of the solution are examined more thoroughly including snippets of the actual code.

5.1 Classes

5.1.1 Project.cs

Variables

- **Public static string folderFullPath** – Full path to the projects folder.
- **Public static string projectName** – Just the name.
- **Public static List<string> fastas** – List of imported FASTA files.
- **Public static List<string> ErrorLog** – Error list for the current project and current session, entire log is available in the form of a file.

Note: The static modifier for these variables is chosen because it is highly undesirable for them to exist in more instances since we always work with only one project at the time.

- **Public static class ItemsToCompute** – Contains extension classes ItemsSelectedToCompute, ItemsToComputeGC and ItemsToComputeANITETRA. These custom collection objects handle various collections of sequences or sequence pairs.

Methods

5.1.1.1.1 Public static bool NewProject(string name)

This method handles creating the project and its folder and file structure. It also loads the global settings from the registry, or their default values using the GlobalSettings class and calls the SaveProject method.

5.1.1.1.2 Public static bool SaveProject()

This method saves the project data into the project master file (project_name.nsat). These data comprise of the number of imported sequences and their full paths. It also makes sure that at the time of saving, the sequence files exist and are accessible. In case of errors, it deletes entries for the missing files and throws appropriate errors and warnings.

5.1.1.1.3 Public static bool OpenProject(string path)

Opens an existing project by parsing the master file and checking availability of the imported sequences. If a project is already loaded, it first saves it and then loads the new one.

5.1.1.1.4 Public static bool ImporFastasFF(string [ ] names)

Gets file paths in an array from OpenFileDialog, checks their availability and then validates them by using ValidateFile. It also calls SaveProject.

5.1.1.1.5 Public static bool ValidateFile(string path)

FASTA validation has to be implemented here since there are no fast and reliable libraries available for C# that would handle this. Loading different file formats according to the file extension is solved via the OpenFileDialog parameters.

```
public static bool ValidateFile(string path)
        {
            StreamReader reader = new StreamReader(path);
            string line = reader.ReadLine();
            line = line.TrimStart();
            for (int i = 0; i < 100; i++)
            {
                if (line == "") line = reader.ReadLine();
                else if (line.StartsWith(">"))
                {
                    break;
                }
            }
            line = line.TrimStart();
            if (!line.StartsWith(">"))
            {
                return false;
            }
            line = reader.ReadLine();
            while (line != null)
            {
                line = line.TrimStart();
                if (line.StartsWith(">")) line = reader.ReadLine();
                else
                {
                    line = line.ToLower();
                    foreach (char i in line)
                    {


                        if (i == 'g' || i == 'c' || i == 's' || i == 't' || i == 'a'
|| i == 'w' || i == 'r' || i == 'y' || i == 'm' || i == 'k' || i == 'h' || i == 'b' ||
i == 'v' || i == 'd' || i == 'n' || i == '\r' || i == ' ' || i == '\t')
                        {
                            line = reader.ReadLine();
                        }
                        else return false;
                    }

                }
               line = reader.ReadLine();
            }
            return true;
        }
```

While the above snippet might look convoluted and overly complicated for a simple validation, it is justified. To illustrate we will break the method down:

- Full path of the FASTA file in question serves as the input; this file is read line-by-line by the StreamReader. To get rid of accidental whitespaces and/or tabs, we first trim the start.

- Then, we proceed to start searching for the header while being liberal and allowing up to 99 empty lines at the start of the file. If we don't find the header on the 99th line, we consider the file invalid.
- After finding the header, we proceed to read the file line by line till the end or until we encounter an invalid character in the sequence.
- We also take into account the possibility of a multi FASTA file by trimming every line and checking its start for the header symbol before analysing it character-by-character.
- We do not, however, allow for a header to be on the same line as a part of the sequence, since this is no longer only formal format violation, but instead a functional one, with potential to skew the results.

5.1.2 ANI.cs

Variables

This class houses no variables outside of its methods.

Methods

5.1.2.1.1 Public static int computeANIb (List<Project.ItemsSelectedToCompute> pairsToCompute)

The name of this method is perhaps somewhat misleading, since it does not actually compute the ANIb value; instead, it facilitates this by calling the ANI function which does the actual computation and passing it the appropriate file paths.

It performs checks for file existence and if there is a problem, it will alter the item list accordingly and throw appropriate warnings and errors.

It also performs (prior to the ANIb calls) multithreaded chopping of all sequence files used in the computation to the 1020 base fragments using the ChopQuery() mehod call.

The reasoning for the return value of this function being an integer is the fact that it basically outputs error codes and the code calling this method then evaluates what kind of error has been encountered. This is ultimately a question of granularity; the method could have outputted a string, but in the authors opinion, this would be excessive and unnecessary.

5.1.2.1.2 Private static void ChopQuery(string PathQ)

This method takes care of chopping the query into the 1020 base long chunks, while not omitting the last fragment that is generally of shorter length.

The file is first read line-by-line to remove the headers, the resulting sequence is then examined on per-character basis and spaces, tabs and newlines are removed (counting on the fact that the file had to be validated earlier). Then a string builder is used to build the resulting sequence, which should result in improved performance compared to regular expressions. Finally, the chunks are created. This is handled by accessing the string by index (in practice string is a char [ ]). Each chunk is saved into the *project_name/temp/file_name/*folder. FASTA header of each chunk consists of its original header and the number of the chunk in question. Filename is then simply *chunk_no.fsa*.

5.1.2.1.3 Private static bool ANIb(string PathA, string PathB)

This is the most important method in this class. It takes two sequences and calculates the ANIb value for them.

```
private static bool ANIb(string PathA, string PathB)
        {
            if (File.Exists(PathA))
            {
                if (File.Exists(PathB))
                {
                    int ChunkNo = 0;
                    double ani = 0;//sum of alignment scores from all chunks
                    string dir = Project.folderFullPath + "\\temp\\ani\\" +
Path.GetFileNameWithoutExtension(PathB);
                    List<string> files=Directory.GetFiles(dir, "*.fsa",
SearchOption.TopDirectoryOnly).ToList();
                    Parallel.ForEach(files, (file) => //multihtreaded blasting much
performance such wow
                    {// Start the new process.
                        Process p = new Process();
                        // Redire   ct the output stream from shell to SO
                        p.StartInfo.UseShellExecute = false;
                        p.StartInfo.RedirectStandardOutput = true;
                        p.StartInfo.RedirectStandardError = true;
                        p.StartInfo.CreateNoWindow = true;
                        p.StartInfo.FileName = GlobalSettings.BlastnExecutablePath;
                        p.StartInfo.Arguments =
                        " -outfmt \"6 length qlen nident pident mismatch gapopen\" " +
//custom format separated by tab with only the results relevant for this use, not that
not all fields are currently used
                        "-subject " + Path.GetFullPath(PathA) + " " +
                        "-query " + Path.GetFullPath(file) + " " +
                        "-penalty -1 "+
                        "-gapopen 5 " +
                        "-gapextend 2 " +
                        "-xdrop_gap_final 150 " +//parameters specified in methodology
                        "-evalue 1e-15 " + //denoise (expected coincidental matches)
"-max_target_seqs 1 "+ //we want just the top (best) alignment
                        " " + "-dust no"; //remove regions of low complexity
```

There are several parts worth going over.

We can notice that a parallel for each loop is used - contents of this loop are equal to blasting and processing the result of one fragment of the query sequence vs the subject sequence. The shell execute is disabled and the output is redirected to the StandardOutput, which we can parse. Blastn path is fetched and start arguments are specified. These include specifying the custom format of the blast output format 6. In this, we specify the fields we are interested in. Note that some of the values are not currently used, however they are relevant for planned graphical representation of the alignments. Resulting output are desired values separated by tabs, which is a simple and easy to parse format.We then read the output using StreamReader, wait for the process to exit and examine the exit code.

Here we first examine the code with which the blastn process ends. Anything else than a 0 is an error.

```
if (p.ExitCode != 0) //exit code !=0 is a BLAST error
                        {
                            string error = "\"" + p.StartInfo.FileName + "\" " +
p.StartInfo.Arguments + "\n" + p.StandardError.ReadToEnd();
                            MainWindow.main.Status = error;
                            Project.ErrorLog.Add(error);

                        }

                        else
                        {//parse output


                                string[] cells = output.Split('\t');
                                if (cells.Length == 6)
                                {
                                double TotalIdentity = 0;
                                double coverage = 0;
                                double AlignLength;
                                double matched;
                                AlignLength = Double.Parse(cells[0]);
                                matched = Double.Parse(cells[2]);
                                double FragLentgth;
                                FragLentgth = Double.Parse(cells[1]);
                                if (AlignLength >= 1020) coverage = 1;
                                else  coverage =   AlignLength / FragLentgth;
                                    if (coverage >= 0.700000)
                                    {
                                        TotalIdentity =
((matched/FragLentgth)*coverage);
                                        if (TotalIdentity > 0.300000)
                                        {

                                            InterlockedAddDouble(ref ani,
TotalIdentity);
                                        Interlocked.Increment(ref ChunkNo);
                                        }
```

In case of successful calculation (exit code==0), we will analyze the output. Here we parse the cells from the output and if the requirements outlined in the ANI chapter are met we will use it for the ANIb value calculation.

What is interesting from a programmer standpoint here is the Interlocking and thread safety consideration. We need to change two variables outside of the parallel loop. ChunkNo can be incremented by the already implemented Interlocked.Increment method. The ani variable which sums all the identity values is trickier. There is the Interlocked.Add method; however, this only works for integers. This led the author to implement a custom method.

InterlockedAddDouble(ref double refLocation, double add)

```
private static void InterlockedAddDouble(ref double refLocation, double add)
        {//custom method for thread safe double addition
            double startVal = refLocation;

            while(true)
            {
                double current = startVal;
                double newVal = startVal + add;
                startVal = Interlocked.CompareExchange(ref refLocation, newVal,
current);
                if (startVal == current) break;
            }

        }
```

The main purpose of this method is to allow change to the referenced value only when it hasn't been changed during the process of addition. This is realized by the while(true) infinite loop and the Interclocked.CompareExchange, that exchanges the value only when the reference did not change and the escape from the loop is solved by the if condition and the break command.

5.1.2.1.4 Public static bool GenerateCSVAni(string file)

This and the **public static bool GenerateCSVAniAvg(string file**) methods are used to generate the csv result matrices. These methods use typical nested foreach loops with alterations to present either bi-directional or averaged ANIb values.

5.1.3 GCC.CS

Variables

No global variables are present in this class.

Methods

5.1.3.1.1 Public static bool computeGCcontent(string path)

Using a simple foreach cycle with if conditions for different characters, this method calculates overall sequence length, number of fragments (by summing up the number of lines starting with the header character *">"*, number of individual A, T, G and C bases, number of N bases, number of other ambiguity symbols and the %GC content.

Most notable here is that unlike the JSpecies and JSpeciesWS the %GC calculation is implemented correctly therefore only decidedly G or C and decidedly A or T bases are considered.

5.1.3.1.2 Public static bool generateCSVGc(string file)

The method used for generating the result matrix, standard foreach cycle taking all the files in the result folder and presenting the contained information in the csv format.

5.1.4 TETRA.cs

This class was implemented last, with the old implementation being completely scrapped in favor of the new and improved one. This is a good example of clean and efficient programming, unlike some other parts of this project. Multithreading is handled optimally here and overall there is little to improve further.

Variables

No global variables in the TETRA class.

Methods

5.1.4.1.1 Private static Dictionary<string,double> computeZscores(string path)

This method takes the path of a sequence file, loads it and computes z scores for every one of the 256 tetranucleotides using the maximum order Markov model.

Detailed breakdown with source code bellow:

```
private static Dictionary<string, double> computeZscores(string path)
        {
            //first load the sequence, remove the headers and keep only G,C,A and T
bases while concatenaning with the reverse complement
            try
            {
                StringBuilder sb = new StringBuilder();
                StringBuilder sbC = new StringBuilder();
                StreamReader reader = new StreamReader(path);
                string line = reader.ReadLine();
                while (!String.IsNullOrEmpty(line))
                {
                    if (!line.StartsWith(">"))
                    {
                        line = line.ToUpper();
                        foreach (char c in line)
                        {
                            if (c == 'A')
                            {
                                sbC.Append('T');
                                sb.Append('A');
                            }
                            else if (c == 'T')
                            {
                                sbC.Append('A');
                                sb.Append('T');
                            }
                            else if (c == 'G')
                            {
                                sbC.Append('C');
                                sb.Append('G');
                            }
                            else if (c == 'C')
                            {
                                sbC.Append('G');
                                sb.Append('C');
                            }
                        }
                    }
                    line = reader.ReadLine();
                }
                string compl = sbC.ToString();
                char[] array = new char[compl.Length];
                int forward = 0;
                for (int i = compl.Length - 1; i >= 0; i--)
                {
                    array[forward++] = compl[i];
                }
                compl = new string(array);
                string seq = sb.ToString() + compl;
```

The first part of the code disregards the header lines, then all the ambiguous bases, since those cannot be used to calculate tetranucleotide frequencies. StringBuilder is used for performance reasons. Also, by using two, we can construct the complementary strand at the same time. All we have to do then is reverse the complementary strand and concatenate the two strings. The char array method used for the reversal should provide the best performance.

Next, it was necessary to create dictionaries for observed and expected tetranucleotide frequencies and derived z-scores along with the observed di/tri nucleotide frequencies. These are keyed by a string of the oligonucleotide itself and double datatype stores the frequency value. Filling for the keys was handled via nested foreach cycles providing variations with repetitions from A, C, G and T characters.

```
char[] n4 = { 'A', 'C', 'G', 'T' };
                char[] tetra = new char[4];
                Dictionary<string, double> observed4 = new Dictionary<string,
double>(); //fill the list with all 256 tetranucleotide combinations as keys
                Dictionary<string, double> expected4 = new Dictionary<string,
double>();
                Dictionary<string, double> zscores = new Dictionary<string, double>();
                foreach (char n1 in n4)
                {
                    tetra[0] = n1;

                    foreach (char ntwo in n4)
                    {
                        tetra[1] = ntwo;
                        foreach (char nthree in n4)
                        {
                            tetra[2] = nthree;
                            foreach (char nfour in n4)
                            {
                                tetra[3] = nfour;
                                observed4.Add(new string(tetra), 0.0);
                                expected4.Add(new string(tetra), 0.0);
                                zscores.Add(new string(tetra), 0.0);
                            }
                        }
                    }


                }
                char[] n3 = { 'A', 'C', 'G', 'T' };
                char[] tri = new char[3];
                Dictionary<string, double> expected3 = new Dictionary<string,
double>();
                Dictionary<string, double> observed3 = new Dictionary<string,
double>();
                //fill the list with all trinucleotide combinations as keys
                foreach (char n1 in n3)
                {
                    tri[0] = n1;
                    foreach (char ntwo in n3)
                    {
                        tri[1] = ntwo;


                        foreach (char nthree in n3)
                        {
                            tri[2] = nthree;
                            observed3.Add(new string(tri), 0.0);
                            expected3.Add(new string(tri), 0.0);

                        }

                    }


                }
                char[] n2 = { 'A', 'C', 'G', 'T' };
                char[] di = new char[2];
                Dictionary<string, double> observed2 = new Dictionary<string,
double>(); //fill the list with all dinucleotide combinations as keys
                Dictionary<string, double> expected2 = new Dictionary<string,
double>();
```

Filling the values for observed frequencies is facilitated by a for cycle accessing four, three and two-character long substrings from the sequence analyzed. Next, the expected frequencies and later, the derived z-scores are computed based on the formulas detailed in the theoretical portion of this thesis.

```
                foreach (char n1 in n2)
                {
                    di[0] = n1;
                    foreach (char ntwo in n2)
                    {
                        di[1] = ntwo;
                        observed2.Add(new string(di), 0.0);
                        expected2.Add(new string(di), 0.0);

                    }


                }
                int length = seq.Length;
                for (int i = 0; i < length - 3; i++)
                {//tetranucleotide observed frenquencies
                    observed4[seq.Substring(i, 4)] += 1;
                }
                for (int i = 0; i < length - 2; i++)
                {//trinucleotide observed frenquencies
                    observed3[seq.Substring(i, 3)] += 1;
                }
                for (int i = 0; i < length - 1; i++)
                {//dinucleotide observed frenquencies
                    observed2[seq.Substring(i, 2)] += 1;
                }
}
                foreach (string tet in observed4.Keys)
                {//Fill the expected tetranucleotide frenquencies
                    expected4[tet] = (observed3[tet.Substring(0, 3)] * obser-
ved3[tet.Substring(1, 3)]) / observed2[tet.Substring(1, 2)];
                }

                foreach (string tet in observed4.Keys)
                {//compute zscores for each tetranucleotide
                    zscores[tet] =(observed4[tet]- expected4[tet])/ (expected4[tet] *
((observed2[tet.Substring(1, 2)] - observed3[tet.Substring(0, 3)]) * (obser-
ved2[tet.Substring(1, 2)] - observed3[tet.Substring(1, 3)]))) / (observed2[tet.Sub-
string(1, 2)] * observed2[tet.Substring(1, 2)]);
                }                 return zscores;
            }
            catch
            {
                return null;
            }
```

The method returns a dictionary of z-scores keyed by the corresponding tetranucleotides for the given sequence.

5.1.4.1.2 Pirvate static double ComputePearsons(Dictionary<string, double> zscoresA, Dictionary<string, double> zscoresB)

This is a relatively simple method that takes two z-score dictionaries for the two sequences that are being compared and using foreach loop computes the Pearson`s correlation coefficient in this context referred to as Tetranucleotide frequency correlation coefficient.

5.1.4.1.3 Public static bool computeTetra(List<Project.ItemsSelectedToCompute> FastaPairs)

This method takes a custom collection implemented in the Project class as the input argument. This is a collection of unique pairs selected by the user in the UI, unlike the collection passed to ANIb methods, it is not bi-directional (instead of A-B and B-A pairs there is only one of the two).

```
  public static bool computeTetra(List<Project.ItemsSelectedToCompute> FastaPairs)
        {//Compute Tetra score(Pearsons cc of z scores from seq a and seq b) using
implemented functions
         //The way we store information is a dictionary with a given fastas filename
as a key and dictionary of its zscores as a value
            try
            {

                List<string> uniqueFiles = new List<string>();

                foreach (var pair in FastaPairs)
                {

                    if (!uniqueFiles.Contains(pair.FullPath))
uniqueFiles.Add(pair.FullPath);
                    if (!uniqueFiles.Contains(pair.FullPathB))
uniqueFiles.Add(pair.FullPathB);




                }
                ConcurrentDictionary<string, Dictionary<string, double>> Zscores = new
ConcurrentDictionary<string, Dictionary<string, double>>();
                Parallel.ForEach(uniqueFiles, (file) =>
                {
                    Zscores.TryAdd(file, computeZscores(file));

                });

                Parallel.ForEach(FastaPairs, (pair) =>
                {
                    double tetra = ComputePearsons(Zscores[pair.FullPath],
Zscores[pair.FullPathB]);
                    if (!File.Exists(Project.folderFullPath + "\\TETRA\\" +
Path.GetFileNameWithoutExtension(pair.FullPath) + "_vs_" +
Path.GetFileNameWithoutExtension(pair.FullPathB) + ".tet"))
                    {
                        File.WriteAllText(Project.folderFullPath + "\\TETRA\\" +
Path.GetFileNameWithoutExtension(pair.FullPath) + "_vs_" +
Path.GetFileNameWithoutExtension(pair.FullPathB) + ".tet", tetra.ToString("F6"));
                    }
                });
                return true;
            }
            catch { return false; }
        }
```

It is good to note the use of multithreading in this method; first, using a ConcurrentDictionary to store the z-scores so they can be computed in parallel for unique sequences and access to the storing structure remains thread safe. Second, for correlation coefficient calculation.

5.1.5 GlobalSettings.cs

Variables

Public static string WorkspaceFolder

Public static string BlastnExecutablePath

Self-explanatory.

Methods

5.1.5.1.1 Public static bool LoadGlobals()

Loads the two path values from the corresponding registry entries. In case there are no values saved in the registry it saves the default ones. Enviroment.SpecialFolder.MyDocuments is used for stability.

5.1.5.1.2 Public static bool SaveGlobals()

Saves the current values to the registry.

5.1.6 MainWindow.xaml.cs

Variables

Internal static MainWindow main – Enables access to MainWindow controls from different classes if needed.

Internal string status – Using data binding, the status variable is bound to the contents of the ListBox lbStatus which is a control in the MainWindow showing the current status of the program.

Methods

5.1.6.1.1 Enable/disable methods

These methods simply enable and disable different sets of controls. They are called from different methods at different times. The main idea is for the user not to be able to access controls that would not work at the time anyway. For example, trying to import files while no project is loaded, or trying to compute ANIb values without having sequences (or just one) imported.

5.1.6.1.2 Public MainWindow()

The entry point. It initializes the application components, loads global settings, sets up the databinding on status and makes sure all controls, except Project->Open, Project->Close and Tools->Settings are disabled.

5.1.6.1.3 Control click methods

The rest of this class is comprised of the click methods handled through RoutedEvent arguments. These methods are called when the user clicks the corresponding control. They ensure that proper functions with proper inputs are called; they also take care of enabling/disabling the controls by calling the associated methods and output appropriate status, warning and error messages. Examining them any closer is not needed.

5.1.7 SelectionWindow.cs and GCSelectionWindow.cs

These two classes are owned by the main. They serve as controls for pair/sequence selection for the implemented calculations.

They make use of the custom collections implemented in the Project class. Another worthwhile mention goes to the use of databinding to generate dynamic controls from the bound collection.

SelectionWindow controls the ANIb and TETRA calculation, since these two parameters are usually computed together. However, there is of course an option to select one or the other or both for each pair using the appropriate checkboxes.

GCSelectionWindow controls the %GC content calculation along with other sequence parameters such as length, number of each base type, number of fragments, number of unknown bases and number of ambiguous bases. It is handled in a similar fashion to the SelcetionWindow.

5.1.8 Other windows

While the about window was simply not yet implemented, since it would serve no practical purpose, the SettingsWindow class is very simple, serving to pass and verify the path variables between the program and the user. This is handled by calling the GlobalSettings class.

6 User manual

This manual aims to familiarize the user with the programs functionality and its GUI. It explains how to perform all of the supported actions.

6.1 Program installation

Initiate the installation by navigating to the **NSAT_install** folder on the included optical disk. Alternatively it is possible to compile the source code from the folder **NSAT_source** on the same disk. It is however recommended to use the first option, since the application will check for updates that can improve its functionality and make the user experience more pleasant. After the installation is complete, the program will launch automatically. In case your computer does not have Microsoft .NET Framework installed, it should automatically prompt you to download and install it. However, an installation file is included in the **NSAT_install** folder.

6.2 BLAST+ installation

Installation file for the latest version of BLAST+ is included on the optical drive inside the **NSAT_install** folder. It can also be downloaded from:

ftp://ftp.ncbi.nlm.nih.gov/blast/executables/blast+/

The NSAT application was tested using BLAST+ in the 2.7.1 stable version, so it is highly recommended to download this exact version.

The installation wizard should lead you through the setup process with no problems; however, make sure to make a note of the installation folder used.

6.3 Step-by-step user guide

The following guide familiarizes you with all the available functions and options supported by the NSAT application. This guide assumes you managed to compile or install the program and it is currently running.

1. On the first launch, the program should notify you that the path to blastn.exe is not set. Confirm this dialog.
2. Next, locate the horizontal menu bar at the top of the window and hover your cursor over the **Tools** menu item.

3. A drop down menu will appear, from that hover over the **Settings** option and click on it.
4. The **Settings** window will appear on top of the NSAT window. Here you need to set the path to the blastn.exe file for the ANIb calculation to function. You will find this file in the **bin** folder inside you NCBI BLAST+ installation directory.
5. To input the path, you can either use the text box adjacent to the „BLASTn executable location" label, or by pressing the **… (three dots)** button next to the aforementioned text box and navigating to the file using Windows explorer.

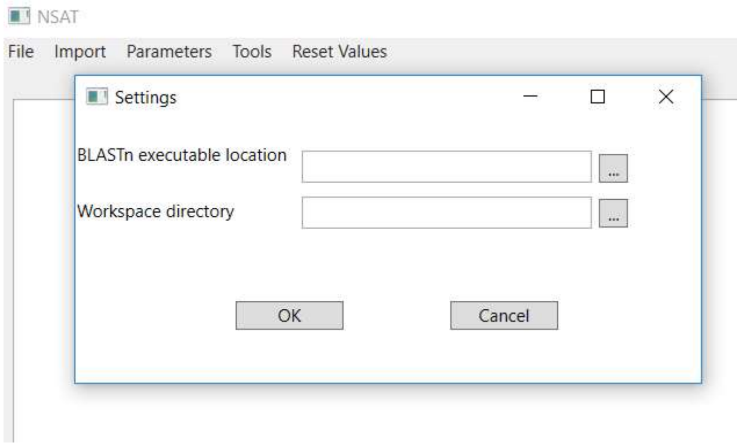


Figure 10 NSAT Settings

6. You can also change the workspace location here; however, it is recommended to keep it as is. The default location for the workspace folder is //DOCUMENTS//NSAT.
7. Next, you should create your first project.
8. Select the **File** menu item and in the dropdown menu, click on **New Project.**

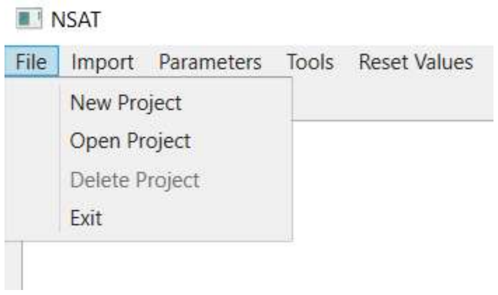


Figure 11 NSAT File menu

9. The **New Project** window will appear. In the text box **Project name,** fill in the desired name for your project. If you input name of an existing project, the project creation will fail and you will be notified by an error message in the status bar at the bottom of the NSAT window.

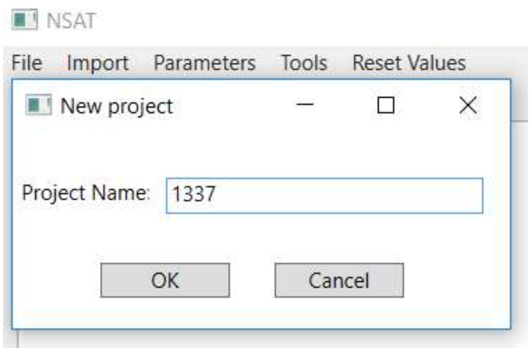
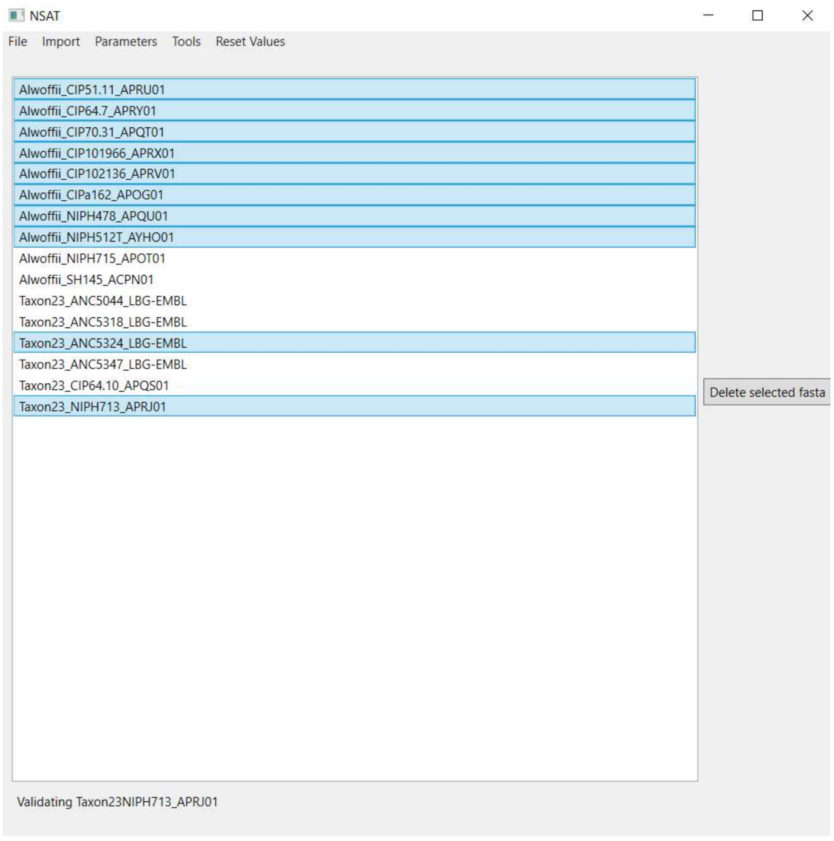


Figure 12 NSAT New project

10. You can also open an existing project in this state by clicking on **Open project** in the **File** submenu.
11. After the project is successfully created or opened, you should be notified in the status bar. Note that if you are loading an existing project, all the previously imported files will be loaded with, if they are still accessible. This also means they will be validated again, which can take a couple seconds.
    Once a project is loaded you, can also choose to delete it. This is done by selecting the **Delete project** option from the **File** menu. This will delete the projects folder and all of its contents, which means all of the computed results for this project will be deleted unless you saved a result matrix outside of the project folder. You will be asked to confirm your choice before the project is irreversibly removed.
12. Now you are free to import sequence files to your project. Select the **Import** menu item and **Fastas from files** from the submenu. You can now load .fasta, .fsa, .fas or even .txt files (Note that they need to have a proper fasta header and a nucleotide sequence as the content). Multiselect is enabled, which means you can select multiple files and load them at once.
    If you wish to remove some of the files from your project, you can do so by selecting them in the list box inside the NSAT window where they are shown. Single selection is realized by clicking on the item, while multiple selection can be achieved by either holding down the control key and clicking or by holding down the shift key and click-dragging your cursor over the items.

File validation can take a few seconds depending on the number of sequences being loaded, computer speed and sequence length. After this is done, you are free to select and calculate the desired parameters.



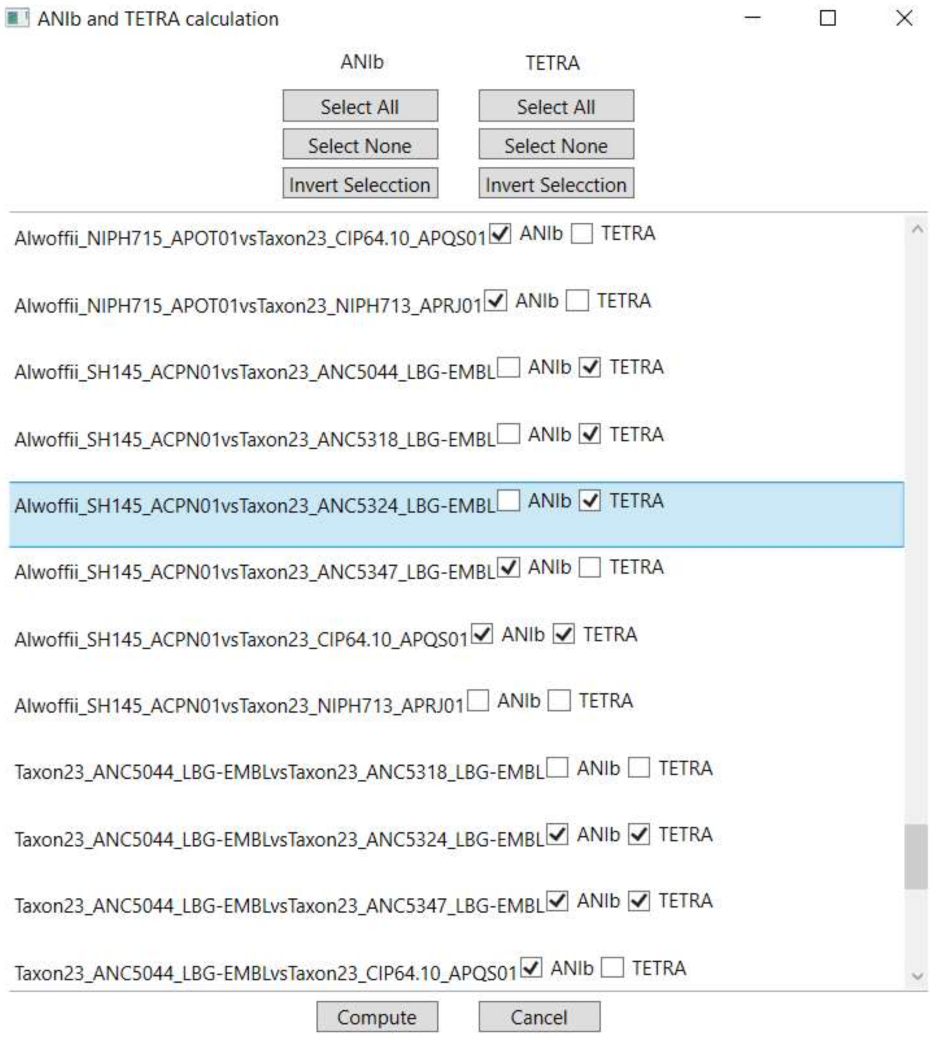
Alwoffii_CIP51.11_APRU01
Alwoffii_CIP64.7_APRY01
Alwoffii_CIP70.31_APQT01
Alwoffii_CIP101966_APRX01
Alwoffii_CIP102136_APRV01
Alwoffii_CIPa162_APOG01
Alwoffii_NIPH478_APQU01
Alwoffii_NIPH512T_AYHO01
Alwoffii_NIPH715_APOT01
Alwoffii_SH145_ACPN01
Taxon23_ANC5044_LBG-EMBL
Taxon23_ANC5318_LBG-EMBL
Taxon23_ANC5324_LBG-EMBL
Taxon23_ANC5347_LBG-EMBL
Taxon23_CIP64.10_APQS01
Taxon23_NIPH713_APRJ01

Validating Taxon23NIPH713_APRJ01

Figure 13 NSAT loaded project

13. Select the **Parameters** menu item; in its submenu, select **Compute** and from its list choose either **ANIb\TETRA** or **GC Content** depending on which parameter you want to compute.
14. After you make yout choice, a computational window will open. In case of the **ANIb/TETRA** parameters, it will contain all the possible unique pairs (variations without repetition) created from the imported sequences and check boxes for both of the parameters for each pair. In case of the **GC content,** it will contain all of the unique sequences each with a single check box.
15. Make your selection of pairs/sequences and desired parameters by checking the appropriate check boxes. The **Select all, Select none** and **Invert selection** controls available in both windows can help you when it comes to bigger datasets.

16. The computation is initiated by clicking the **Compute** button. **Caution:** While the TETRA and %GC parameter calculations are extremely fast and will take only seconds (TETRA takes slightly longer than %GC), the ANIb calculation takes minutes to hours in larger datasets. It is recommended to use only a few sequences for testing purposes in case of this parameter.


ANIb and TETRA calculation

ANIb: Select All | Select None | Invert Seleccion
TETRA: Select All | Select None | Invert Seleccion

Alwoffii_NIPH715_APOT01vsTaxon23_CIP64.10_APQS01 ✓ ANIb ☐ TETRA
Alwoffii_NIPH715_APOT01vsTaxon23_NIPH713_APRJ01 ✓ ANIb ☐ TETRA
Alwoffii_SH145_ACPN01vsTaxon23_ANC5044_LBG-EMBL ☐ ANIb ✓ TETRA
Alwoffii_SH145_ACPN01vsTaxon23_ANC5318_LBG-EMBL ☐ ANIb ✓ TETRA
Alwoffii_SH145_ACPN01vsTaxon23_ANC5324_LBG-EMBL ☐ ANIb ✓ TETRA
Alwoffii_SH145_ACPN01vsTaxon23_ANC5347_LBG-EMBL ✓ ANIb ☐ TETRA
Alwoffii_SH145_ACPN01vsTaxon23_CIP64.10_APQS01 ✓ ANIb ✓ TETRA
Alwoffii_SH145_ACPN01vsTaxon23_NIPH713_APRJ01 ☐ ANIb ☐ TETRA
Taxon23_ANC5044_LBG-EMBLvsTaxon23_ANC5318_LBG-EMBL ☐ ANIb ☐ TETRA
Taxon23_ANC5044_LBG-EMBLvsTaxon23_ANC5324_LBG-EMBL ✓ ANIb ✓ TETRA
Taxon23_ANC5044_LBG-EMBLvsTaxon23_ANC5347_LBG-EMBL ✓ ANIb ✓ TETRA
Taxon23_ANC5044_LBG-EMBLvsTaxon23_CIP64.10_APQS01 ✓ ANIb ☐ TETRA

Compute | Cancel

Figure 14 NSAT computation

17. When the computation is finished, you will be notified by the program.
18. You can now export the results in .csv format by selecting **Parameters->Export CSV** and the appropriate parameter the export options will be active only for the parameters that have been computed for at least one sequence or pair, respectively.
    In case of ANIb, you can pick either full or average matrix. The full matrix resents the bi-directional values for the parameter, however, this is rarely

used for testing purposes. It is better to pick the matrix with averaged pairwise distances.

19. The last option provided is the **Reset results** option. By selecting this item, you can choose a parameter for which you want to delete previously computed results by clicking on it in the sub menu.

20. You can exit the application either by clicking the **X** in the top right hand corner of the NSAT window or by navigation to the **File** menu and selecting the **Exit** option.

21. You do not have to worry about saving your progress. It is done automatically anytime you change a setting, add a sequence file or another result (for a single pair or sequence) is computed.

22. Finally, even by terminating the program in non standard way such as killing the process or shutting down your computer will not cause any problems. You will only lose partially computed results..

7 Results and testing

7.1 Testing

Two separate testing methods (Unit and User testing) have been used.

7.1.1 Unit testing

For data and domain layer testing, unit tests have been created where possible using Microsoft Visual Studio and testing mostly the implemented algorithms with the overall data handling.

<unit test example snipper>

7.1.2 User testing

UI and application testing

For the purposes of presentation layer testing and overall application testing, user tests have been conducted ensuring that the resulting program works as intended.

This was realized by testing all the functions of this program on two separate machines – one running Windows 10 and the other running Windows 7 with both systems having the latest updates installed and using the latest version of NCBI BLAST at the time (version 2.7.1 stable) for ANIb calculation. Multiple tests have been conducted in order to test all of the functionality and different combinations and succession of user actions. Features tested were:

- Launching and quitting the program.
- Creating a new project with a custom name.
- Opening an existing project – not containing any files, containing files, with no results computed, with some results computed, with all results computed.
- Importing files to the project.
- Removing files from the project.
- Changing the workspace folder.
- Changing the blastn executable location.
- Changing the blastn command line application parameters and the effect on ANIb results.
- Forcefully terminating the program while computation is in progress and recovering results.
- Exporting and viewing result files.
- Programs ability to deal with invalid sequence files.

Results accuracy testing

Testing accuracy of the achieved computational results is crucial to ensure that the algorithms have been implemented properly.

Testing sequences were selected as a part of a bigger sequence set used for an article co-authored by the author of this thesis. The article in question deals with taxonomic revision of the *Acinetobacter lwoffii* group. While the article has been already submitted it is now in pre-print and pending review status.

The article in question is titled "Revising the taxonomy of the *Acinetobacter lwoffii* group: the description of *Acinetobacter pseudolwoffii* sp. nov. and emended description of *Acinetobacter lwoffii*" and its pre-print version can be found on the attached compact disc. The article was submitted to the journal "*Systematic and Applied Microbiology*".

All testing for ANIb and TETRA parameters was done on the same set of sequence files. For %GC testing a different, more suitable set of sequences was used and %GC values from the NCBI nucleotide database served as control (since this metric is calculated improperly in both JSPecies and JSpeciesWS).

The tables with actual results used for this testing are attached on the optical disc in the \\Attachements\\Validation\\ folder, since it is pretty much impossible to make them a part of this printed docuent in any satisfactory manner.

8 Discussion

As can be seen from the testing and results section, the application is functional for its purpose, while providing accurate results. There are however some caveats and other things worth noting when it comes to the individual genomic parameters and their results.

8.1 User testing

While the results of user testing are mostly positive, there can still be stability issues in cases of highly non-standard user behaviour.

8.2 Result analysis

8.2.1 ANIb results analysis

We can notice a slight deviation from JSpecies and JSpeciesWS controls in ANIb results. However, this variation can be attributed to a different and newer version of the BLAST algorithm and its heuristic nature. This is supported by the fact that the ANb values obtained by JSpecies and JSpeciesWS deviate from each other as well. Nevertheless, all these deviations are small (<0.1%) and occur only for low ANIb values, thus being insignificant for the purposes of this application.

That being said, further investigation is required and has been initiated by the author in a form of an inquiry to dr. Ramon Rosselló-Móra, the senior author of the JSpecies and JSpeciesWS applications. The reason for this inquiry was the fact the blastn parameters specified in the JSpeciesWS documentation are functional only for the older BLAST algorithm, while JSpeciesWS and our new application uses the new and improved BLAST+. Although, it is possible to "translate" all of the parameters used into their equivalents for the BLAST+, the issue comes with the required parameter "penalty" of -1. Specifying this parameter causes BLAST+ to throw an error forcing the user to specify two more parameters, "gap open penalty" and "gap extension penalty", in a specific range. Because these parameters are not specified in the methodology, it is possible that the two mentioned parameters are the source of the inconsistencies. In the current state those are set to 5 and 2 respectively, mirroring the legacy BLAST default values. The creators of the JSpeciesWS software therefore had to specify these parameters, assuming that they indeed used the penalty of -1, which would lead to results that more closely mirror those obtained with the previously used software. To verify this assumption, ANIb was calculated using the legacy BLAST algorithm and no difference outside the margin of error was found between these values and the values calculated by the original JSpecies. However, using the legacy BLAST version is not recommended as it is inferior in many ways, including performance, to the newer BLAST+, while also being no longer supported. Therefore, our program will continue to use the BLAST+.

8.2.2 %GC results analysis

When it comes to the %GC results, we can see that our results deviate from those produced by both JSpecies and JSpeciesWS in some cases, which should not happen since this is an exact metric, not a heuristic estimation. In this case, the controls are to blame. When checked against the NCBI nucleotide database, the results from both control applications differ at times, while our results are always on point. This indicates that the implementation of this genomic parameter in the two control programs is incorrect and confirms the observation of others that JSpecies does not work properly.

It is likely that the faulty calculation of %GC by JSpeciesWS is caused by improper handling of the many ambiguity symbols that can appear in a FASTA file. Proper implementation should only take into an account G, C , A, T, S and W. What most likely happens (inferred from the fact that when deviating, %GC scores yielded by JSpeciesWS are slightly lower than controls) that the overall sum of decidedly G or C bases in the whole sequence is probably calculated correctly, but as a next step instead of dividing by the sum of decidedly G or C plus the sum of decidedly A or T the G or C sum is divided by sequence length. This then also takes into account all of the other ambiguity symbols such as N or K that do not hold enough information to be decidedly members of either of the pairs and should therefore not be used in the calculation at all. The deviations found for %GC values calculated by JSpecies are too high to be explainable as above and it is possible that this software suffers from another systematic error

8.2.3 TETRA results analysis

Tetranucleotide correlation coefficient results are satisfactory, being identical with the controls. Furthermore, the implementation has been shown to be extremely fast, handily outperforming the original JSpecies program, while in case of the JSpeciesWS such comparison cannot be made since it runes on a remote server.

8.3 Future goals

It is clear from the result analysis, that there is still work to be done on this application.

Firstly, it would be ideal to achieve ANIb results within the margin of error of the JSpeciesWS results, but this depends on the methodology clarification being expected from dr. Ramon Rossello-Mora and his colleagues.

Furthermore, the stability of the program should be tested further, even though it does not currently exhibit any issues in this area.

Another consideration includes a computational speed improvement. In its current state, our application is not making the best use of the modern multithreaded systems for the ANIb calculation. The JSpecies control is often somewhat faster when computing the ANIb parameter - therefore optimizations could definitely be made. Some possible improvements are quite obvious, the biggest bottleneck at the present time seems to be the I/O (storage) performance, this could potentially be alleviated by either using BLAST+ to create a custom database or storing the fragments in a data structure. The author decided against the first option because it adds the need to build the database from all of the fragments for each sequence first, which is not exactly computationally light. The argument against the second possible solution was a fear of oversaturating the memory subsysten on older or less powerful PCs, in retrospect this argument is somewhat flawed and not a real concern outside of very large datasets. Overall, performance optimization for ANIb is one of the most important future goals that will require a considerable time investment and experimentation. Of course, the author`s lack of coding experience somewhat contributes to this problem, so it can be expected that the application will improve with the author's abilities and experience.

Next in line would be improving the graphical user interface. While the presently used is definitely serviceable, it leaves more to be desired from a practical standpoint such as:

- Improving the selection management.
- Adding support for keyboard shortcuts.
- Adding the possibility to view results in the GUI itself without any need for third party applications.

Improving the GUI from an aestethic persepctive is extremely low on the list of priotities, since it would not serve any practical purpose.

From the functional perspective, there are some other genomic parameters worth exploring and possibly adding such as oligonucleotide correlation (five and six), Average Amino Acid Identity (AAI) (Konstantinidis, 2005b) and the Microbial Species Identifier (MiSI) (Varghese, 2015).

8.4 Practical usability

With the above-mentioned limitations in mind, the resulting application is a usable tool for in silico sequence-based DNA comparison. As such, it will be deployed in the following weeks as a main sequence comparison tool in the Laboratory of Bacterial Generics at the National Health Institute and used for planned extensive taxonomical studies.

9 Conclusion

Overall, all the goals of this thesis have been fulfilled with some minor caveats and a few compromises as summarized below and analyzed in the “Discussion” chapter. The best way to gauge success or lack thereof when it comes to this thesis is to recap the goals outlined in the “Goals” chapter and try to evaluate if and into what extent have they been fulfilled.

9.1 Goal fulfillment evaluation analysis

- To review theory and research behind selected in silico sequence based genomic parameters, their predecessors and alternatives as well as their in vitro counterparts and predecessors to gauge their usability, advantages and limitations.
    - ✓ Many papers and articles relevant to the study subject have been perused, although not all of them are cited. The most relevant articles are listed in the “Literature” section.
- To gain deep understanding of mentioned algorithms and their inner workings.
    - ✓ The author managed to gain deep understanding of the methods involved and the theory behind them. The most relevant methods and algorithms are described in the “Current state” chapter.
- Based on this review, to decide which parameters should be prioritised and which (if any) should be omitted.
    - ✓ The most important parameter is ANIb as it is widely regarded as a taxonomic gold standard.

      %GC is considered as having the second priority. While it is a very simple measurement, it is widely used and takes virtually no computational time.

      TETRA has shown some potential, while it is not as widely adopted as the previous ones and when it comes to conclusiveness of its results it lies somewhere between the two, it has proven to work extremely fast, in practical terms its speed is very close to that of the %GC. This parameter was included mainly because it has a potential to determine whether or not ANIb should be calculated.
- To analyse the need for configurability to provide only relevant options while avoiding unnecessary options and confusing the user.
    - ✓ All of the algorithms are rigidly defined. Therefore, providing any configuration options that would affect results to the user would be

counterproductive. The initial idea to offer some of the BLAST+ command line parameters as configurable was implemented but later scrapped since it ultimately made little to no sense. It is not among the goals for this application to serve as a BLAST+ GUI. So in the end, only configuration options present in the program are related to paths to the blastn executable and option to change the workspace folder location.

- To design a project-based saving system that will ensure data are preservation and accessibility while enabling the program to recover from non-standard termination without data corruption and loss of already computed results.
    - ✓ The way this application works pretty much ensures that the second an option is changed, or a result is computed it saves it into a file (or a registry key) which is simple but effective. The only data that are lost are the partial calculations if the program exited in a non-standard manner during the process.
- To design the application structure while taking speed and efficiency into consideration, ideally using multiple processing threads.
    - ✓ This is a partial success. While performance was always considered when writing the code and multithreading was implemented where possible, there are definitely improvements to be made. Especially when it comes to the ANIb parameter.
- To design the application architecture that will fit outlined goals while being compatible with personal computers based on x86 processor architecture and running the Microsoft Windows operating system.
    - ✓ The application runs well under Microsoft Windows on standard personal computers.
- To implement the solution based on the outlined design and its functional and non-functional specifications.
    - ✓ The implementation fulfils the vast majority of the specified requirements.
- To document the implementation in a reasonable degree of detail (i.e. make sure that the source code snippets are not the dominant part of this thesis's practical part).
    - ✓ This has been done in the "Implementation" chapter of this thesis.
- To create a simple user manual that explains the application functionality from a practical standpoint.
    - ✓ This has been done in the "User manual" chapter.

9.1.1 The missing dendrogram

While it might have not been noticed sadly the function to export a dendrogram is currently missing from the program. While UPGMA clustering and Newick format export have been implemented, unfortunately the author did no test the implementation thoroughly until the day before deadline when it became clear that it is not functioning properly, the problem lies either in the implementation of the clustering algorithm or in building the Newick file. Sadly, not enough time to diagnose and repair the problem was left therefore this function has been temporarily removed. The author however believes that it is a minor bug and the dendrogram support will be restored shortly after turning this thesis in.

9.2 Future outlook

As for what the future holds for this program, that has been proposed in the "Future goals" section of the "Discussion" chapter. The very distant, somewhat optimistic and definitely ambitious endgame would be for the NSAT application to become a capable and efficient DNA analysis tool supporting a wide range of parameters and functionality while remaining accessible to biologists who are just regular users, not programmers or bioinformaticians. The author's idea is to make in silico sequence as an area of bioinformatics more accessible.

9.3 License

Usage of this software and its source code is governed by the MIT license.

Literature used

A. PRAY, Leslie, 2008. Discovery of DNA Structure and Function: Watson and Crick. *Nature* [online]. **2008**(1), 1-1 [cit. 2018-05-16]. Dostupné z: https://www.nature.com/scitable/nated/article?action=showContentInPopup&contentPK=397

ALTSCHUL, Stephen F. a Bruce W. ERICKSON, 1986. Optimal sequence alignment using affine gap costs. *Bulletin of Mathematical Biology* [online]. **48**(5-6), 603-616 [cit. 2018-05-17]. DOI: 10.1007/BF02462326. ISSN 0092-8240. Dostupné z: http://link.springer.com/10.1007/BF02462326

ALTSCHUL, Stephen F., Warren GISH, Webb MILLER, Eugene W. MYERS a David J. LIPMAN, 1990. Basic local alignment search tool. *Journal of Molecular Biology* [online]. **215**(3), 403-410 [cit. 2018-05-17]. DOI: 10.1016/S0022-2836(05)80360-2. ISSN 00222836. Dostupné z: http://linkinghub.elsevier.com/retrieve/pii/S0022283605803602

A-T DNA base pair, 2007. In: *Commons.wikimedia.org* [online]. online: Wiki [cit. 2018-08-17]. Dostupné z: https://commons.wikimedia.org/wiki/File:AT_DNA_base_pair.svg

Bioinformatics and molecular evolution, 2005. 2005. Oxford: Blackwell Publishing. ISBN 14-051-3802-5.

BLAST® Command Line Applications User Manual [online], 2008. **2008**(1), 1-200 [cit. 2018-05-17]. Dostupné z: https://www.ncbi.nlm.nih.gov/books/NBK279682/

BOUVIER, Thierry a Paul A DEL GIORGIO, 2003. Factors influencing the detection of bacterial cells using fluorescence in situ hybridization (FISH): A quantitative review of published reports. *FEMS Microbiology Ecology* [online]. **44**(1), 3-15 [cit. 2018-05-17]. DOI: 10.1016/S0168-6496(02)00461-0. ISSN 01686496. Dostupné z: https://academic.oup.com/femsec/article-lookup/doi/10.1016/S0168-6496(02)00461-0

Central dogma of molecular biology, 2013. In: *MMG 233 2013 Genetics & Genomics Wiki* [online]. [cit. 2018-05-17]. Dostupné z: https://vignette.wikia.nocookie.net/mmg-233-2014-genetics-genomics/images/6/66/Central-dogma.png/revision/latest?cb=20140901203620

COMPEAU, Phillip a Pavel PEVZNER, 2015. *Bioinformatics algorithms: an active learning approach*. 2nd Edition. La Jolla, CA: Active Learning Publishers. ISBN 978-099-0374-619.

DNA 3′ 5′ end [online], 2013. In: . [cit. 2018-05-16]. Dostupné z: https://dlc.dcccd.edu/images/biology/lesson3/phosphodiester_linkages_model.jpg

DNA structure detail, b.r. In: *Https://ptetchem.wordpress.com* [online]. [cit. 2018-05-16]. Dostupné z: http://3.bp.blogspot.com/-K4VKM6UVP3A/Tke2H-FtlxI/AAAAAAAABM4/y8LRnu9xf8Y/s1600/DNA+backbone1.jpg

EDITED BY MARKUS SCHMIDT., , 2012. *Synthetic biology industrial and environmental applications*. 1. Hoboken: Wiley-Blackwell. ISBN 978-352-7659-265.

FASTA format - NCBI [online], b.r. [cit. 2018-05-16]. Dostupné z: https://blast.ncbi.nlm.nih.gov/Blast.cgi?CMD=Web&PAGE_TYPE=BlastDocs&DOC_TYPE=BlastHelp

G-C DNA bae pair, 2010. In: *Commons.wikimedia.org* [online]. online: Wiki [cit. 2018-08-17]. Dostupné z: https://commons.wikimedia.org/wiki/File:GC_DNA_base_pair.svg

HEATHER, James M. a Benjamin CHAIN, 2016. The sequence of sequencers: The history of sequencing DNA. *Genomics* [online]. **107**(1), 1-8 [cit. 2018-05-16]. DOI: 10.1016/j.ygeno.2015.11.003. ISSN 08887543. Dostupné z: http://linkinghub.elsevier.com/retrieve/pii/S0888754315300410

IUPAC-IUB COMM. ON BIOCHEM. NOMENCL, , 2002. Abbreviations and symbols for nucleic acids, polynucleotides, and their constituents. *Biochemistry* [online]. **9**(20), 4022-4027 [cit. 2018-05-16]. DOI: 10.1021/bi00822a023. ISSN 0006-2960. Dostupné z: http://pubs.acs.org/doi/abs/10.1021/bi00822a023

Jspecies, 2009. *Jspecies* [online]. imedea: imedea [cit. 2018-05-16]. Dostupné z: http://imedea.uib-csic.es/jspecies

JspeciesWS [online], 2015. Ribohost: Ribohost [cit. 2018-05-16]. Dostupné z: http://jspecies.ribohost.com/jspeciesws

KIMSEY, Isaac J., Eric S. SZYMANSKI, Walter J. ZAHURANCIK et al., 2018. Dynamic basis for dG•dT misincorporation via tautomerization and ionization. *Nature* [online]. **554**(7691), 195-201 [cit. 2018-08-15]. DOI: 10.1038/nature25487. ISSN 0028-0836. Dostupné z: http://www.nature.com/doifinder/10.1038/nature25487

KLAPPENBACH, Joel, Johan GORIS, Peter VANDAMME, Tom COENYE, Konstantinos KONSTANTINIDIS a James TIEDJE, 2007. DNA–DNA hybridization values and their relationship to whole-genome sequence similarities. *International Journal of Systematic and Evolutionary Microbiology* [online]. **57**(1), 81-91 [cit. 2018-05-16]. ISSN 1466-5026. Dostupné z: http://ijs.microbiologyresearch.org/content/journal/ijsem/10.1099/ijs.0.64483-0#tab2

KONSTANTINIDIS, K. T. a J. M. TIEDJE, 2005b. Towards a Genome-Based Taxonomy for Prokaryotes. *Journal of Bacteriology* [online]. **187**(18), 6258-6264 [cit. 2018-08-13]. DOI: 10.1128/JB.187.18.6258-6264.2005. ISSN 0021-9193. Dostupné z: http://jb.asm.org/cgi/doi/10.1128/JB.187.18.6258-6264.2005

KONSTANTINIDIS, K. a J. TIEDJE, 2005a. Genomic insights that advance the species definition for prokaryotes. *Proceedings of the National Academy of Sciences* [online]. **102**(7), 2567-2572 [cit. 2018-05-16]. DOI: 10.1073/pnas.0409727102. ISSN 0027-8424. Dostupné z: http://www.pnas.org/cgi/doi/10.1073/pnas.0409727102

LAND, Miriam, Loren HAUSER, Se-Ran JUN et al., 2015. *Insights from 20 years of bacterial genome sequencing* [online]. **15**(2), 141-161 [cit. 2018-05-16]. DOI: 10.1007/s10142-015-0433-4. ISSN 1438-793X. Dostupné z: http://link.springer.com/10.1007/s10142-015-0433-4

LOCEY, Kenneth J. a Jay T. LENNON, 2016. Scaling laws predict global microbial diversity. *Proceedings of the National Academy of Sciences* [online]. **113**(21), 5970-5975 [cit. 2018-05-16]. DOI: 10.1073/pnas.1521291113. ISSN 0027-8424. Dostupné z: http://www.pnas.org/lookup/doi/10.1073/pnas.1521291113

MCLEAN, Phil, 2004. BLAST: Basic Local Alignment Search Tool. *Colorado State University lecture* [online]. 1-17 [cit. 2018-05-17]. Dostupné z: https://www.ndsu.edu/pubweb/~mcclean/plsc411/Blast-explanation-lecture-and-overhead.pdf

MOORE, W., E. STACKEBRANDT, O. KANDLER et al., 1987. Report of the Ad Hoc Committee on Reconciliation of Approaches to Bacterial Systematics. *International Journal of Systematic and Evolutionary Microbiology* [online]. **37**(4), 463-464 [cit. 2018-05-16]. DOI: 10.1099/00207713-37-4-463. ISSN 1466-5026. Dostupné z: http://www.microbiologyresearch.org/content/journal/ijsem/10.1099/00207713-37-4-463

MYERS, Eugene W. a Webb MILLER, 1988. Optimal alignments in linear space. *Bioinformatics* [online]. **4**(1), 11-17 [cit. 2018-05-17]. DOI: 10.1093/bioinformatics/4.1.11. ISSN 1367-4803. Dostupné z: https://academic.oup.com/bioinformatics/article-lookup/doi/10.1093/bioinformatics/4.1.11

NAGEL, Christian, 2009. *C# 2008: programujeme profesionálně*. 2008. Brno: Computer Press. Programujeme profesionálně. ISBN 978-80-251-2401-7.

NEEDLEMAN, Saul B. a Christian D. WUNSCH, 1970. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of Molecular Biology* [online]. **48**(3), 443-453 [cit. 2018-07-20]. DOI:

10.1016/0022-2836(70)90057-4. ISSN 00222836. Dostupné z: http://linkinghub.elsevier.com/retrieve/pii/0022283670900574

NISHIDA, Hiromi, 2012. Comparative Analyses of Base Compositions, DNA Sizes, and Dinucleotide Frequency Profiles in Archaeal and Bacterial Chromosomes and Plasmids. *International Journal of Evolutionary Biology* [online]. **2012**, 1-5 [cit. 2018-05-17]. DOI: 10.1155/2012/342482. ISSN 2090-8032. Dostupné z: https://www.hindawi.com/archive/2012/342482/

PEARSON, William R., 2002. Selecting the Right Similarity-Scoring Matrix. *Current Protocols in Bioinformatics* [online]. Hoboken, NJ, USA, **1**(1), 351-359 [cit. 2018-08-15]. DOI: 10.1002/0471250953.bi0305s43. ISBN 9780471250951. Dostupné z: http://doi.wiley.com/10.1002/0471250953.bi0305s43

RICHTER, M. a R. ROSSELLO-MORA, 2009. Shifting the genomic gold standard for the prokaryotic species definition. *Proceedings of the National Academy of Sciences* [online]. **106**(45), 19126-19131 [cit. 2018-05-16]. DOI: 10.1073/pnas.0409727102. ISSN 0027-8424. Dostupné z: http://www.pnas.org/cgi/doi/10.1073/pnas.0906412106

ROSSELLÓ-MÓRA, Ramon a Rudolf AMANN, 2015. Past and future species definitions for Bacteria and Archaea. *Systematic and Applied Microbiology* [online]. **38**(4), 209-216 [cit. 2018-05-16]. DOI: 10.1016/j.syapm.2015.02.001. ISSN 07232020. Dostupné z: http://linkinghub.elsevier.com/retrieve/pii/S0723202015000223

ROSSELLÓ-MÓRA, Ramon, Mercedes URDIAIN a Arantxa LÓPEZ-LÓPEZ, 2011. DNA–DNA Hybridization. *Taxonomy of Prokaryotes*. Elsevier, 325-347. Methods in Microbiology. DOI: 10.1016/B978-0-12-387730-7.00015-2. ISBN 9780123877307. Dostupné také z: http://linkinghub.elsevier.com/retrieve/pii/B9780123877307000152

SCHBATH, SOPHIE, BERNARD PRUM a ELISABETH DE TURCKHEIM, 1995. Exceptional Motifs in Different Markov Chain Models for a Statistical Analysis of DNA Sequences. *Journal of Computational Biology* [online]. **2**(3), 417-437 [cit. 2018-05-17]. DOI: 10.1089/cmb.1995.2.417. ISSN 1066-5277. Dostupné z: http://www.liebertonline.com/doi/abs/10.1089/cmb.1995.2.417

SMITH, T.F. a M.S. WATERMAN, 1981. Identification of common molecular subsequences. *Journal of Molecular Biology* [online]. **1981**(1), 195-197 [cit. 2018-05-17]. DOI: 10.1016/0022-2836(81)90087-5. ISSN 00222836. Dostupné z: https://www.sciencedirect.com/science/article/pii/0022283681900875?via%3Dihub

STACKEBRANDT, E. a B. GOEBEL, 1994. Taxonomic Note: A Place for DNA-DNA Reassociation and 16S rRNA Sequence Analysis in the Present Species Definition in Bacteriology. *International Journal of Systematic and Evolutionary Microbiology* [online]. **44**(4), 846-849 [cit. 2018-05-16]. DOI: 10.1099/00207713-44-4-846. ISSN

1466-5026. Dostupné z: http://www.microbiologyresearch.org/content/journal/ijsem/10.1099/00207713-44-4-846

TEELING, Hanno, Anke MEYERDIERKS a BAUER, 2004. *Environmental Microbiology* [online]. **6**(9) [cit. 2018-05-17]. DOI: 10.1111/j.1462-2920.2004.00624.x. ISSN 1462-2912.

VARGHESE, Neha J., Supratim MUKHERJEE, Natalia IVANOVA, Konstantinos T. KONSTANTINIDIS, Kostas MAVROMMATIS, Nikos C. KYRPIDES a Amrita PATI, 2015. Microbial species delineation using whole genome sequences. *Nucleic Acids Research* [online]. **43**(14), 6761-6771 [cit. 2018-08-13]. DOI: 10.1093/nar/gkv657. ISSN 0305-1048. Dostupné z: https://academic.oup.com/nar/article-lookup/doi/10.1093/nar/gkv657

ZVELEBIL, Marketa J. a Jeremy O. BAUM, 2008. *Understanding bioinformatics*. 1. New York: Garland Science. ISBN 08-153-4024-9.