UNIVERSITY OF ZAGREB
FACULTY OF ELECTRICAL ENGINEERING AND COMPUTING

MASTER THESIS num. 1573

Protein Database Search Using Partial Order Alignment

Petar Žuljević

Zagreb, July 2018.

Umjesto ove stranice umetnite izvornik Vašeg rada.
Kako biste uklonili ovu stranicu, obrišite naredbu `\izvornik`.

CONTENTS

1. Introduction

Proteins are large macromolecules that consist of long chains of amino acid residues. Alphabet symbols to represent them belong to "ACDEFGHIKLMNPQRSTVWY" set of letters (20 standard amino acids). The sequence of amino acid residues is determined by genes (defined with genetic code). The role of proteins is multiple, it ranges from support of DNA replication, signal transduction to provide catalyst functions for the enzymes.

Protein database is a set of string sequences which are usually easily accessible and available on the Internet from the sources like NCBI, UniProt/SwissProt etc. Those databases are being updated on a daily basis by dedicated domain experts. Some of entries in the database are manually annotated while others did not go through the review and are just automatically annotated. Currently, the number of stored protein reference sequences and the whole database is growing continuously which makes protein database search problem even more hard. Efficient way to search such big databases is still open question.

Analyzing protein sequences can happen today via process called alignment or other exact methods. Homology denotes that two sequences are originating from the same ancestor, so finding homology between e.g. human and mouse would enable to apply knowledge about one organism to the other one by figuring out the function of the shared gene. If certain protein sequence has unknown origin, the main goal is to compare it with the set of known sequences and infer the common ancestor. For the known sequences we are able to construct evolutionary tree (phylogenetic tree, Figure 1.1).

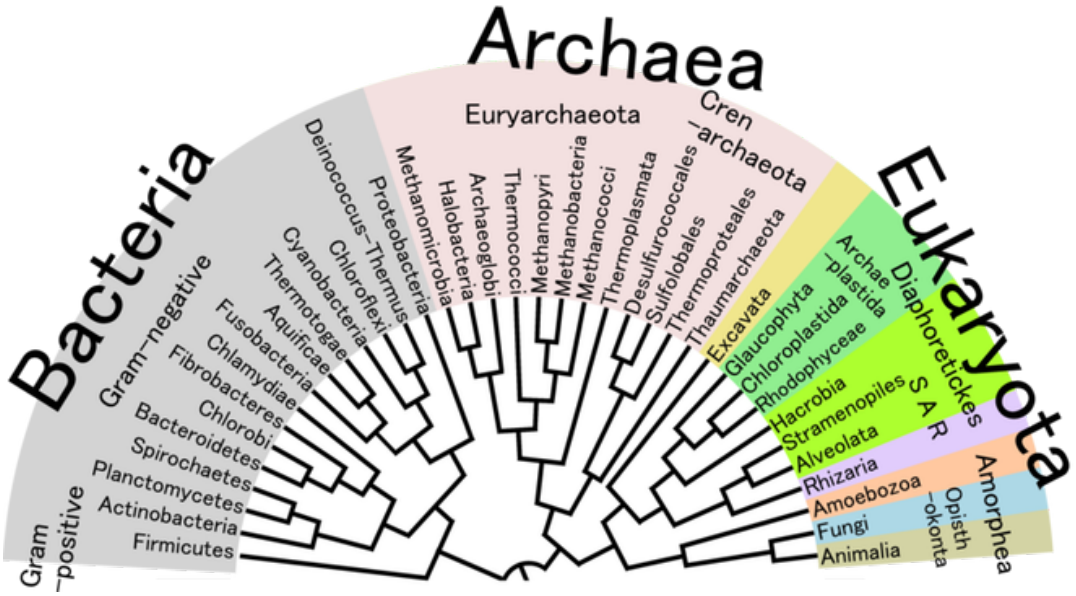

Figure 1.1: Phylogenetic tree example (Moosmosis, 2018)

Structure of the thesis

In the second chapter, we will describe protein database search approach in general, approach selected in this thesis and basic mathematical terms that have been used. Third chapter will be focused on analyzing obtained results of various experiments. Fourth chapter will provide more details on technical design of the solution. Thesis is ending with the conclusion and the list of used literature.

2. The most common approaches and the algorithm developed in this thesis

2.1. Problem formulation

The main goal of sequence alignment algorithms is to come up with the estimate or exact measure how similar two or multiple sequences are. There exist two types of algorithms (exact and heuristic ones). Exact algorithms usually rely on dynamic programming which is too expensive given how large current databases are. Second type is heuristic approach which is much faster, but it does not guarantee to find an optimal solution.

Generally, the common idea is to extract features which can be expressed in a mathematical form for the set of given known and labeled protein sequences and then do the same for the unknown sequence. Based on the appropriate comparisons we should be able to come up with the claim "Sequence X is likely part of group Y".

2.2. Overview of the existing algorithms

The following is the list of existing experimental algorithm methods which are primarily based on the results of experiments in the laboratory as opposed to algorithmic approaches which are primarily analyzing data structure of protein sequence strings.

2.2.1. Experimental methods

Mixture-Spectrum Partitioning using Libraries of Identified Tandem mass spectra permits untargeted and sensitive peptide identification in data-independent acquisition (DIA) data. MSPLIT-DIA is a spectral matching tool that uses spectrum projections to match library spectra to each DIA spectrum. This method also evaluates the similarity of the matched peaks between library spectra and multiplexed spectra across multiple consecutive DIA spectra (Wang et al., 2015).

Informed-Proteomics allows top-down data analysis. Informed-Proteomics is composed of a liquid chromatography mass spectrometry (LC-MS) feature-finding algorithm, a database-search algorithm, and an interactive results viewer. It is used to proceed sensitive and comprehensive high-throughput analysis of complex mixtures of intact proteins. This tool is able to identify differently expressed LC-MS features and proteoforms from breast cancer samples (Park et al., 2017).

RAId-DbS assigns accurate spectrum-specific statistical significance to peptide hits. RAId-DbS is a peptide identification software that takes into account the integrated information of annotated single amino acid polymorphisms (SAPs) and/or post-translational modifications (PTMs) and diseases while performing peptide searches. It offers flexibility for users to add SAPs/PTMs information to various proteins (Alves et al., 2010).

2.2.2. Algorithmic methods

BLASTX Searches protein database using a translated nucleotide query. BLASTX is a BLAST search application that compares the six-frame conceptual translation products of a nucleotide query sequence (both strands) against a protein sequence database. This application can also work in Blast2Sequences mode and can send BLAST searches over the network to public NCBI server if desired (National Center for Biotechnology Information, 2008).

DIAMOND An alignment tool for aligning short DNA sequencing reads to a protein reference database such as NCBI non-redundant (NCBI-nr). On Illumina reads of length 100-150bp, in fast mode, DIAMOND is about 20,000 times faster than BLASTX, while reporting about 80-90% of all matches that BLASTX finds, with an e-value of at most 1e-5. In sensitive mode, DIAMOND is about 2,500 times faster than BLASTX, finding more than 94% of all matches (Buchfink et al., 2014).

SWORD - Smith Waterman On Reduced Database Combines a heuristic and an exact approach. It is aimed to replace BLAST, as it is faster between 8 and 16 times and has better or comparable sensitivity. The second main advantage is that it produces guaranteed optimal alignments. We also presented a faster version of SWORD which sensitivity drops significantly on higher e-values depending on dataset used. Its primary use is to retrieve most similar sequences, those with small e-values, up to 68 times faster than BLAST (Rognes, 2011).

Tachyon Quickly connects a query protein sequence to highly similar sequences and selected associated resources. Tachyon identifies closely related protein sequences 200 times

faster than standard BLAST, circumvents this limitation with a reduced database and oligopeptide matching heuristic (Tan et al., 2012).

BLAST (Basic Local Alignment Search Tool) The most common local alignment tool is BLAST (Basic Local Alignment Search Tool) developed by Altschul et al. (1990. J Mol Biol 215:403). The operative phrase in the phrase is local alignment. The BLAST is a set of algorithms that attempt to find a short fragment of a query sequence that aligns perfectly with a fragment of a subject sequence found in a database (Altschul et al., 1990).

2.3. Algorithm developed in this thesis

Algorithm developed in this thesis is based on the idea to build the protein database with extracted subset of features from given input protein sequences and then perform clustering of data with two layered structure where each layer optimizes the speed of search or sensitivity.

For each of the input queries, alignment will be performed with the partially ordered graphs which consist of the most similar sequences. To give more insights into the approach, the following chapters will give an overview of the algorithms, metrics and tools that have been used to achieve aforementioned.

2.3.1. Data compression

The first step of data processing is focused on extracting features from protein sequences and translating them to the feature vector space. Extracting features will be based on K-minimizer or K-maximizer approach in order to compress overall data storage requirements and therefore optimize execution performance since less features means less CPU operations.

Minimizers

To reduce the storage requirements, utilizing the idea of minimizers might be useful. Instead of storing the whole sequence to the memory we can store only certain subset of k-mers. Unique consistent approach across multiple sequences is to select lexicographically the smallest k-mer in the sliding window of size W as on Figure 2.1.

Position	1	2	3	4	5	6	7	1	2	3	4	5	6	7	8	9	10	11	12
Sequence	2	3	1	0	3	4	3	4	2	6	4	7	2	8	1	4	7	5	1
k-mers	2	3	1					4	2	6	4	7	2	8					
with		3	1	0					**2**	**6**	**4**	**7**	**2**	**8**	**1**				
minimizer			1	0	3					6	4	7	2	8	1	4			
in				**0**	**3**	**4**					4	7	2	8	1	4	7		
bold					3	4	3					7	2	8	1	4	7	5	
	(a)							(b)					2	8	1	4	7	5	1

Figure 2.1: Illustration of all k-mers in two windows of sequence as well as their minimizers (Roberts et al., 2004)

This way, comparing the subset of k-mers between sequences is guaranteed to be consistently selected. Those k-mers are called minimizers. Similarily, choosing lexicographically largest k-mers in the sliding window will generate maximizers.

2.3.2. Similarity measure selection

If Clustering is selected as the approach then the key steps are to define measure based on which we perform clustering and similarity match. Measure selection can greatly impact performance and final results. As a two potential domains of selection we will observe Euclidean and non-Euclidean metrics. For the purpose of this thesis, non-Euclidean metric has been selected (Jaccard).

Euclidean similarity

This is the most usual, "natural" and intuitive way of computing a distance between two samples. This distance type is mostly used for data sets that are normalized or without any special distribution problem. For this use case this metric did not represent good fit due the nature of observed problem.

Jaccard Similarity

Jaccard Similarity – which is also referred to as the Tanimoto coefficient, measures similarity in a way that it observes the intersection divided by the union of two sets (Figure 2.2).

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|}$$

Figure 2.2: Jaccard similarity between the two sets A and B

Example

If the group A contains elements a, a, a, b, b, c and the group B contains a, a, b, b, c, c, c. Jaccard coefficient would be intersection (b + c) divided by the union (a + b + c) = 2 / 3. This is perfect match for the problem where we want deduce similarity based on the subset of sequence k-mers matches.

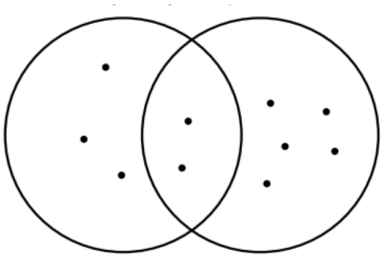

Figure 2.3: Intersection and union of the two sets A and B

2.3.3. Clustering

Basic idea which has been applied is to pull out k-minimizers / k-maximizer and translate given samples into feature vector space. Step after that is to perform similarity analysis based on Jaccard metrics with given known labeled samples and the new unknown ones from the data input.

Tight clustering

Tight clustering (algorithm 1) will quickly provide closest match with the given processed database. The similarity which has been used is a measure which is based on the percentage of matches in feature vector space based on Jaccard.

Algorithm 1 Tight clustering

$W \leftarrow$ set of query protein samples
$tc \leftarrow$ tight clustering threshold
$membership \leftarrow$ uninit
for each: $w \in \mathcal{W}$
  $TC \leftarrow$ set of existing tight clusters
  $maxS \leftarrow$ the max similarity between query sequence *w* and the tight cluster within *TC*
  $l \leftarrow$ tight cluster with similarity *maxS* with the query *w*
  if $maxS \geq tc$ **then**
    $membership \leftarrow l$
  else
    $l2 \leftarrow$ create a new tight cluster
    $membership \leftarrow$ l2

Observations

Say that we have two samples A and B of similar lengths with similar number of k-mers. It's worth to notice how Jaccard metric behaves depending on the number of k-mer matches between feature vectors of A and B. Potential set of values that Jaccard provides will have a integer multiplied with the value 1/N where N is number of extracted k-mers.

This means, having just a few k-mers may not yield a lot of different values for distance metric (so we may have false positives/negatives since many samples would have the same value of the Jaccard similarity just by random chance).

For the samples with shorter length it might be worth to increase sampling rate, while for longer sequences we might use less features.

Loose clustering

The goal of loose clustering (algorithm 2) is to assign less similar sequences (but still greater than loose threshold) to be the part of the larger group which will contain tight clusters within itself. Having this secondary clustering layer enables us to reduce space of search significantly and iterate faster.

Algorithm 2 Loose clustering

$W \leftarrow$ set of query protein samples
$lc \leftarrow$ loose clustering threshold
$membership \leftarrow$ uninit
for each: $w \in \mathcal{W}$
  $LC \leftarrow$ set of existing loose clusters
  $maxS \leftarrow$ the max similarity between query sequence `w` and the loose cluster within *LC*
  $l \leftarrow$ loose cluster with similarity `maxS` with the query `w`
  if $maxS \geq lc$ **then**
    $membership \leftarrow l$
  else
    $l2 \leftarrow$ create a new loose cluster
    $membership \leftarrow$ *l2*

Clustering threshold relaxation

This is heuristic which may help to deal with the case when input data samples are completely random and every new sample does not fit into existing loose / tight clusters. In this case the current solution would generate new loose and tight cluster for each of those samples. This would cause slower execution time due to the fact that increasing number of loose clusters linearly is contributing to overall quadratic execution complexity. Number of loose clusters is expected to grow as square root of the number of read samples.

Threshold relaxation means that each time when this trend is detected, algorithm reduces loose threshold by some constant factor like 0.90, this way it's more likely that new samples will fit eventually some of the existing loose clusters (this trade-off will sacrifice some sensitivity to gain on speed).

Locality-sensitive hashing

Locality-sensitive hashing (LSH) reduces the dimensionality of high-dimensional data. LSH hashes input items so that similar items map to the same buckets with high probability (the number of buckets being much smaller than the universe of possible input items). LSH is trying to maximize the probability of a "collision" for the similar items.

Assume that we want to find the list of similar sequences in the set of size N=1000000 (10^6) elements. Brute force solution would be calculating N^2 Jaccard similarities which would take a several days to complete.

LSH offers the solution for this, find a hash function h such that it satisfies with high probability that:

$$s(A, B) \to 1 \Rightarrow h(A) = h(B)$$
$$s(A, B) \to 0 \Rightarrow h(A) \neq h(B)$$

Since similar sequences are likely to share their hash values (e.g. of the selected k-mers), comparing those elements for which h(A) = h(B) will be sufficient.

One of initial approaches was to use LSH for tight clustering instead of minimizers, then correlation with the number K of hash functions and Jaccard similarity showed that too large parameter K can cause false positives and too long computation time (optimal value turned out to be between 50 and 100). Table 2.1 is showing trend of value changes for 2 random sequences:

Table 2.1: Jaccard similarity for two random sequences when using K hash functions

K	Jaccard similarity
10	0.0931399
20	0.123898
50	0.194438
100	0.296966
500	0.507325

2.4. Partial ordered alignment

POA (partial ordered alignment) is a bit different approach than previously mentioned alignment algorithms and heuristics. POA guarantees that the optimal alignment of each new sequence versus each sequence in the MSA will be considered. Moreover, this algorithm introduces an edit operator, homologous recombination, important for multidomain sequences.

POA's advantages are speed, scalability, sensitivity, and the superior ability to handle branching / indels in the alignment. This algorithm enables the construction of massive and complex alignments. (source: https://omictools.com/partial-order-alignment-tool).

POA Alignment

Given the example of the two protein sequence, there may be some ambiguity in selecting a single, the best alignment between this pair of sequences:

$$> seq1$$
$$YYMYVVVVYYMY$$
$$> seq2$$
$$YYMYPPPPYYMY$$

Alignment 1:

$$YYMY----VVVVYMYM$$
$$YYMYPPPP----YMYM$$

Alignment 2:

$$YYMYVVVV----YYMY$$
$$YYMY----PPPPYYMY$$

Natural way to observe this is via partially ordered alignment graph as on Figure 2.4:

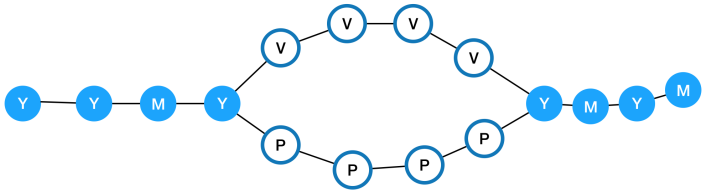

Figure 2.4: POA for selected two sequences

This partial order alignment graph is a different than alignment of strings with some classical dynamic programming approach (e.g. SW / NW). The 'Y' after the fork may be preceded by both 'V' and 'P'. So generalization may happen through extension of SW to observe multiple potential predecessors when looking at the state of each node. This graph is constrained to be a Directed, Acyclic Graph (DAG).

Insertion of aligned sequence

For an example we can observe graph which contains alignment YMYVVPV, and dynamic programming during aligning the sequence YMPVVPYM has provided the new alignment that looks like:

$$\begin{array}{c} YMPVVPYM \\ |\ |\ .\ |\ |\ |\ .\ . \\ YMYVVPV- \end{array}$$

Inserting the new sequence to POA should generate a directed graph that looks like on Figure 2.5:

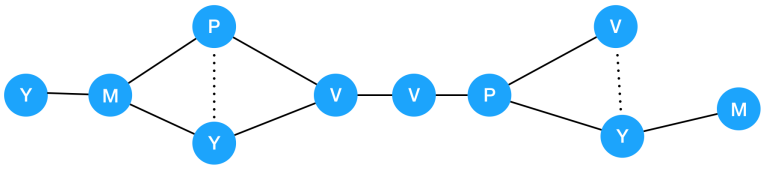

Figure 2.5: POA result after insertion of the sequence to the existing alignment

Application of POA

In the current solution, analysis for the new sequence will happen by aligning it with MSA from the loose cluster that it fits. Loose cluster containing many tight cluster centroids will form MSA.

Algorithm 3 Utilizing POA

$Q \leftarrow$ set of query protein samples
$LC \leftarrow$ set of loose clusters
for each: $q \in \mathcal{Q}$
  $maxS \leftarrow$ the max similarity between query sequence *q* and the loose cluster within *LC*
  $l \leftarrow$ loose cluster with similarity `maxS` with the query `w`
  $TC \leftarrow$ set of tight clusters form `l`
  $poaGraph \leftarrow$ uninit
  $poaGraph \leftarrow$ align sequence *q* to the *poaGraph*
 for each: $t \in \mathcal{T}C$
  $poaGraph \leftarrow$ align protein sequence to the graph which is centroid in `t`

Algorithm 3 will provide good overview about functional/structural relationship of the new sequence with the reference database. SPOA (Vaser et al., 2017) implementation of POA will be used to achieve aforementioned functionalities.

3. Results analysis

3.1. Performance optimizations

Input processing parallelization

After the initial experimentation we have confirmed that it would be beneficial to parallelize input processing. Results below indicate that optimal number of threads is between 50 and 100 resulting in 50x speed up.

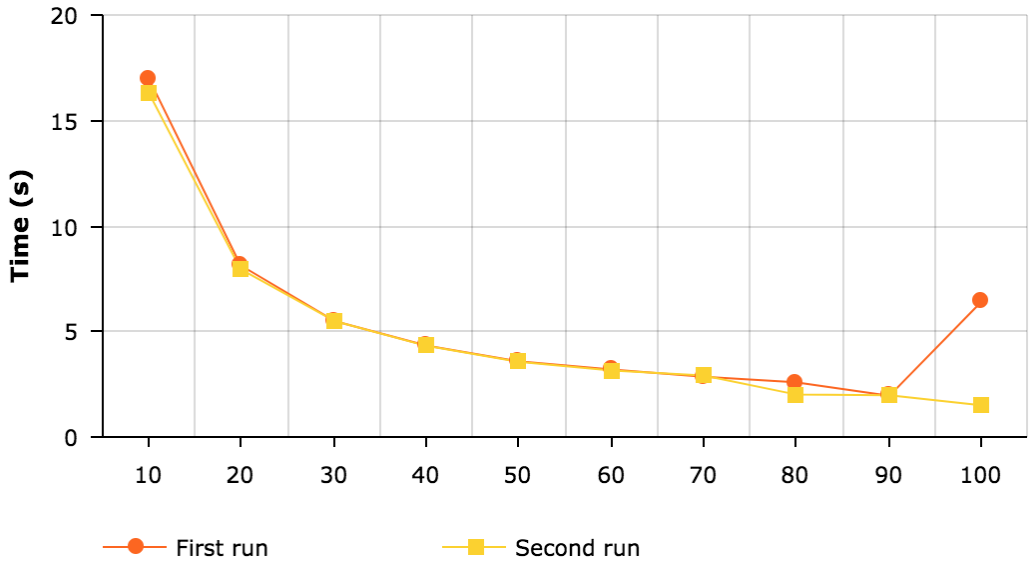

Figure 3.1: IO processing depending on the number of used threads (experiment run 2 times)

For this purpose we have created thread pool with dynamic thread allocation. After processing IO, clustering per length happens and threads can be reused for further processing.

Clustering parallelization based on length buckets

Expectation for the length of the protein sequences is between 300 and 500. This was confirmed by checking Uniprot databases on the Internet and also by running clustering per length locally and observing results. Clustering per length during local runs showed matching with overall UniProt results. This means that it makes sense to utilize 10-20 threads for length buckets since most will be concentrated around values of 350.

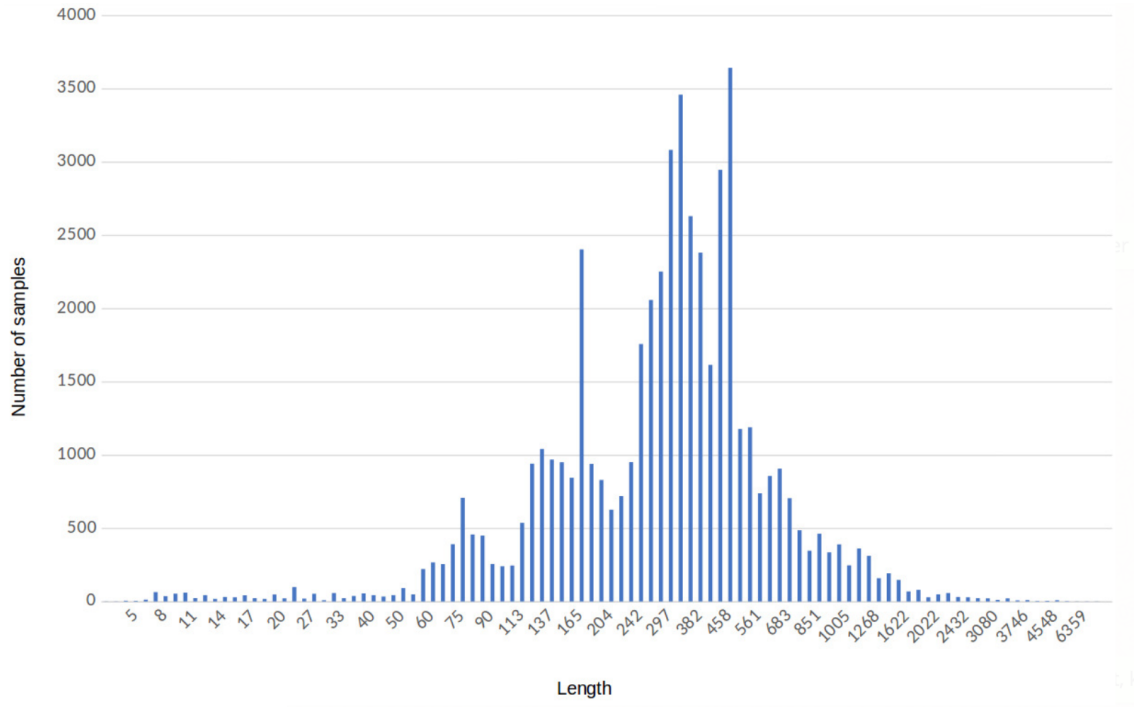

Figure 3.2: Distribution of protein sequences length

3.2. Experiment results

For the purpose of developing better model and getting more understanding, we decided to run clustering on the predefined sets of protein data sequences which were artificially generated by using the script outlined in appendix. Several experiments will be outlined and corresponding results.

As a basic experiment, we've generated 5 similar sequences, where similarity is defined as percentage of different characters on randomly selected positions (e.g. 10% means that 10% of positions have different characters). Name of the sequence denotes that difference, it is in the form: **"C"<seq length>_<percentage of a difference when compared to the seed>**.

Table 3.1: Jaccard similarity for experimental sequences

	C136	C136_0.1	C136_0.5	C136_0.9
C136 (seed)	0	0.65	0.875	1
C136_0.1	0.65	0	0.962963	1
C136_0.5	0.875	0.962963	0	1
C136_0.9	1	1	1	0

Example with distance matrix on table 3.1:

>C136 (seed)

GSADNEQNECDFEWFFDPMSQCCWTDII...LNHKEFIIQCNPPSLPPDRAWQWWQ

>C136_0.1

GSADNEQNECDFEWFFDPTSQCCWTDII...EDHWDRCNLFMMDKWYRHTSDFLP

>C136_0.5

RPADNNQQEADFTWFFGPMAQCCWTDI...IATIDHWDMCNLDWAEWWYRHCLK

>C136_0.9

ERAKDEYNEMARVFTIWPDVQVCTEIAQD...NQQWDSRCMSGMMDTWYPDEDD

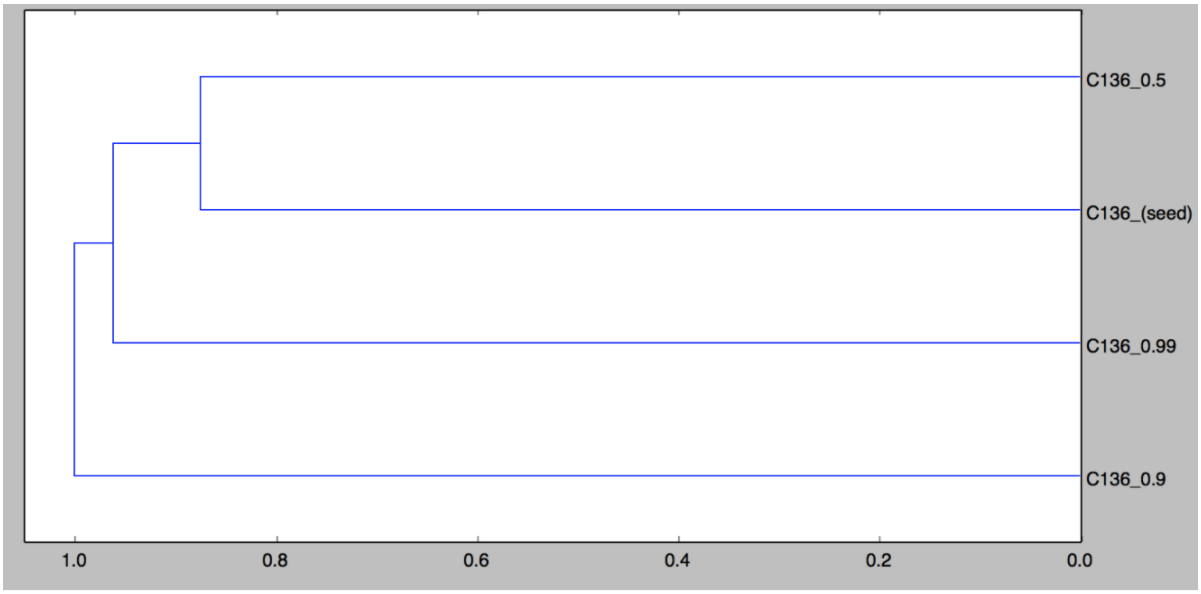

Figure 3.3: Loose clusters when using minimizers (K=3, W=10) and loose threshold=0.2

Explanation: Figure 3.3 is showing loose cluster representatives, we don't see sequences with 1% and 10% of difference because they got included in seed's loose cluster.

3.2.1. Building database

To get more information we decided to extend simulated dataset with more sequences and run more experiments to measure performance, correctness and memory consumption. We grouped those experiments in the following categories: Correctness verification, Generating features and Performance measurement. The experiments provided solid information about overall model design.

Correctness verification experiments

Experiment 1: 10 cluster seeds - no similarities between seeds

Configuration: Using minimizers, K=3, W=10, loose threshold=0.2, tight threshold=0.9

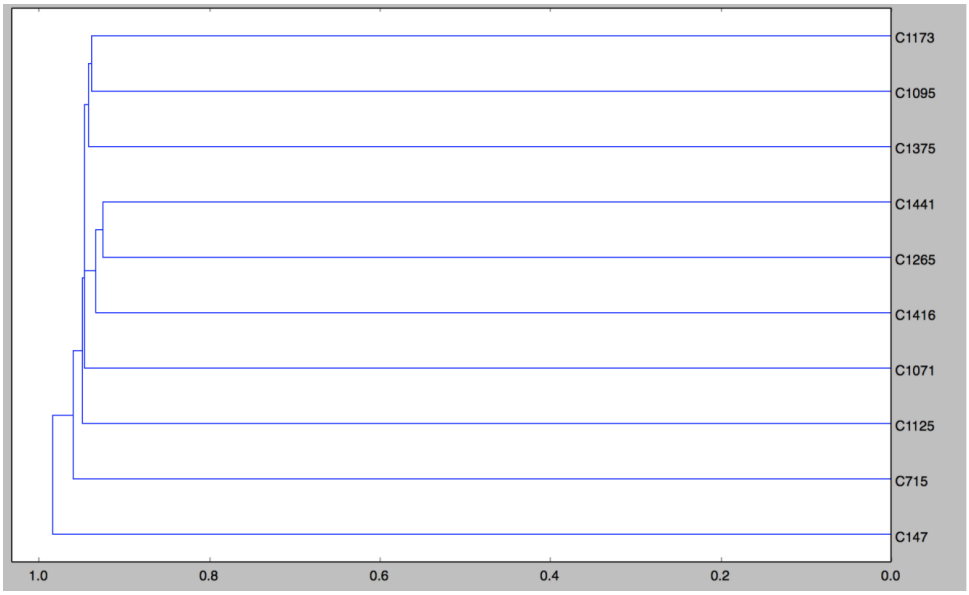

Figure 3.4: Loose clusters with 10 different seed sequences

Result: Total number of loose clusters: 10, total number of tight clusters: 10

Explanation: In Figure 3.4 we can see that similar lengths got in similar loose cluster buckets.

Experiment 2: 10 cluster seeds - similarities 1% / 10% / 25%

Configuration: Using minimizers, K=3, W=10, loose threshold=0.2, tight threshold=0.9

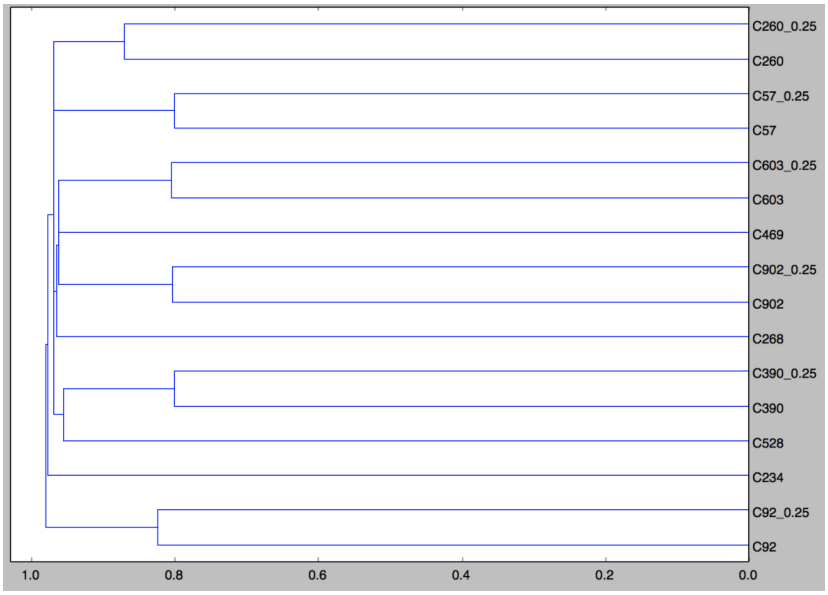

Figure 3.5: Loose clusters with 10 different seed sequences

Result: Total number of loose clusters: 16, total number of tight clusters: 37.

Explanation: In Figure 3.5 we can see that C<XYZ> and C<XYZ>_0.25 ended up really close in diagram which indicates their similarity.

Experiment 3: Modified parameters, 10 cluster seeds - similarities 1% / 10% / 25%

Configuration: Using minimizers, K=3, W=10, loose threshold=**0.5**, tight threshold=0.9

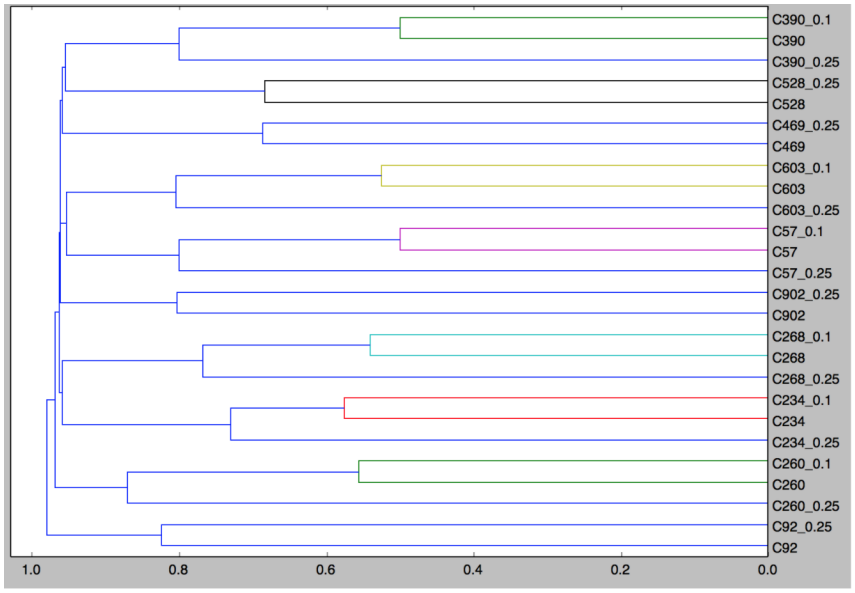

Figure 3.6: Loose clusters with 10 different seed sequences

Result: Total number of loose clusters: 16, total number of tight clusters: 37.

Explanation: In Figure 3.6 we can see that C<XYZ> and C<XYZ>_0.25 ended up really close in diagram which indicates their similarity.

Generating features

In order to have reliable results of clustering it's important to map protein sequences to vector space. In order to achieve that we have several different parameters to experiment with. Most critical parameters are K (length of k-gram) and W (window size). We set **K=3** and experimented with parameter W.

As an dataset to test with, we decided to use 500 protein seed sequences with 1%, 10% and 25% of similarity mismatch per seed. Length is random variable in interval [50, 1500]. This will results with 2000 sequences overall. Given the loose threshold of 0.20 and tight threshold of 0.95, we expect to see 500 loose clusters from seeds and up to 500 more based on 25% similarity mismatch in the worst case.

The following tables 3.2 and 3.3 are showing results of experiments when using different parameters to generate features from samples.

Table 3.2: Number of tight and loose clusters with different window size and minimizers/maximizers selection

	W=10	W=20	W=100	W=1000
Minimizers	loose: 515 tight: 1931	loose: 539 tight: 1846	loose: 605 tight: 1526	loose: 266 tight: 579
Maximizers	loose: 523 tight: 1925	loose: 558 tight: 1847	loose: 707 tight: 1581	loose: 646 tight: 950
Minimizers + Maximizers	loose: 505 tight: 1965	loose: 523 tight: 1924	loose: 625 tight: 1700	loose: 372 tight: 1174

Table 3.3: Number of tight and loose clusters for different W which denotes number of hash functions for LSH

	W=10	W=20	W=100
LSH minhash	loose: 401 tight: 1865	loose: 234 tight: 1835	loose: 44 tight: 1978

Conclusion: Based on the results in 3.2, seems it is sufficient to use (K, W) minimizers with basic hash conversion of string protein k-mer to the integer value. As per 3.3 LSH minhash tends to give false positives with too many hash functions and complexity increases as well. K=3 and W=10 shows as a good pick for minimizers selection.

Core performance measurement

Performance measurement is focused on observing time and memory consumption when dealing with datasets of different sizes (e.g. files of size 1M, 100M and 1GB). Running experiments on the set of those datasets will give insight in the program's behavior. We're using 50 threads to process input file and 20 threads for length clustering while the length was randomly selected from the interval [50, 10000].

Table 3.4: Correlation between performance and number of sequences

	N=50 (1M)	N=5000 (100M)	N=50000 (1GB)
time	0.0041s	27s	55s (real), 5.5m (user)

3.2.2. Search using POA

POA is being used as a final step to align query sequence with the most similar loose clusters of proteins. Once when database is constructed solution will iterate through the query sequences and for each of them find a set of most similar tight clusters representatives from the same loose cluster.

Following experiments are showing how the alignment is being done on POA graph between query sequence to find overall consensus. For the purpose of experiment we've generated 10 seed sequences and their corresponding similar pairs (similarity up to 1%, 10% and 25%) as shown by overall clustering result in Figure 3.7.

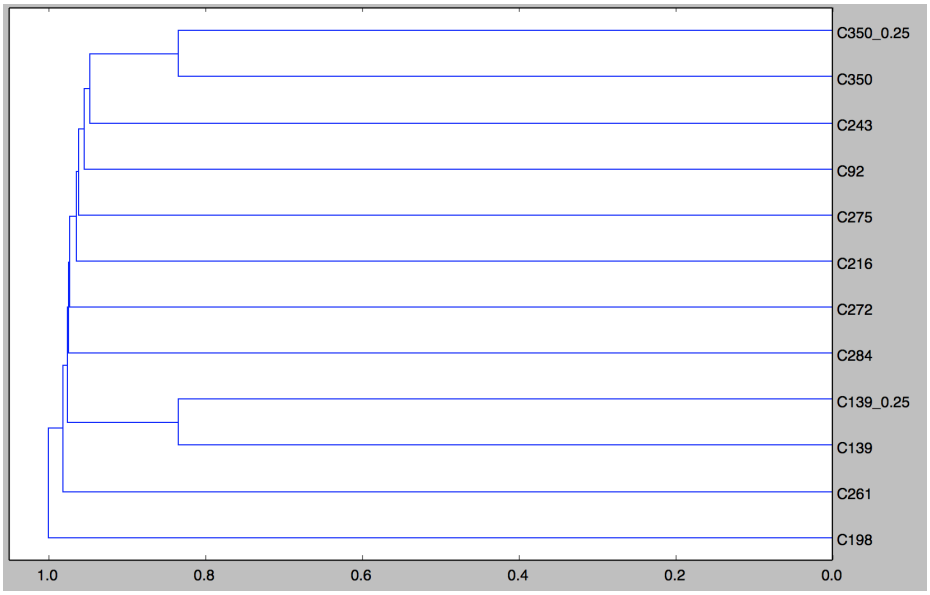

Figure 3.7: Initial database clustering, K=3, W=10, loose_threshold=0.2, tight_threshold=0.9

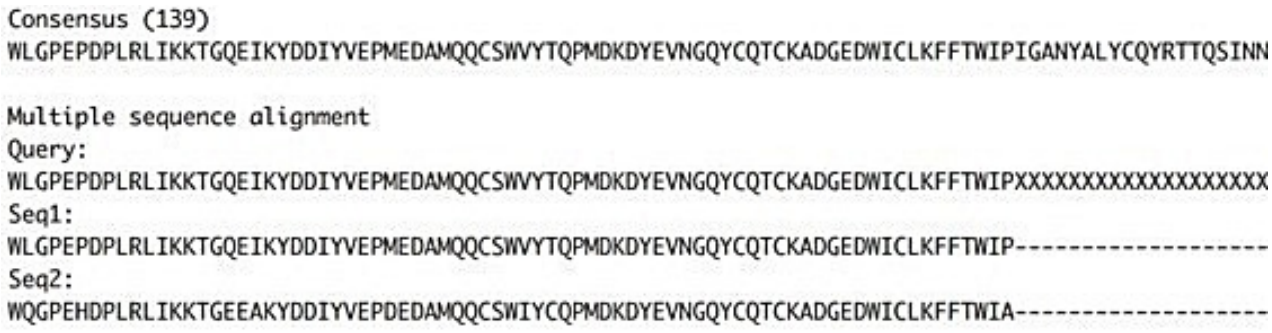

Figure 3.8: Query sequence containing significant substring of tight clusters data, but also having unknown suffix "XXXX..."

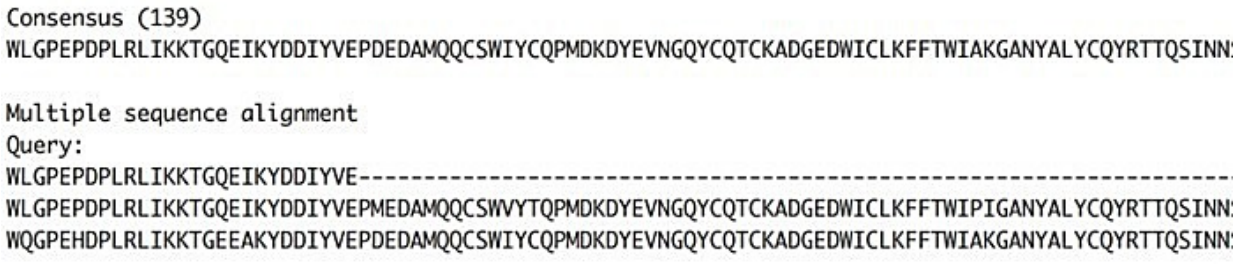

Figure 3.9: Query sequence containing small substring present in both tight cluster represents (seq1 and seq2)

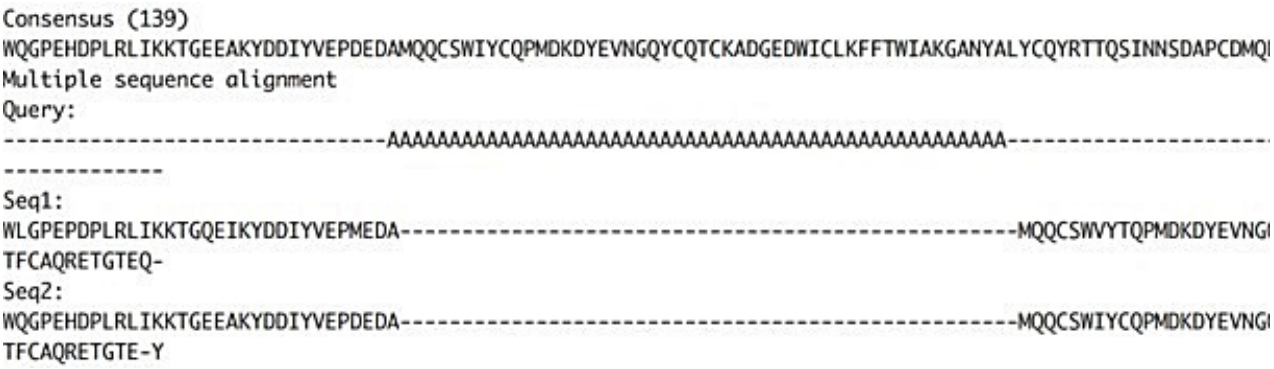

Figure 3.10: Query sequence containing string "AAAA..." which is completely unrelated to seq1 and seq2, tight cluster represents

Figures 3.8, 3.9 and 3.10 are representing the utilization of POA above the various query sequences matched against clustered database. ‘Seq1‘ and ‘Seq2‘ are tight cluster represents of the loose cluster where the query sequence belongs. Figure 3.10 is showing invalid case in theory since the sequence is complete outlier, but algorithm will still try to assign the best possible loose cluster (in this case all clusters are equally good).

3.3. Comparison with other existing tools

In this chapter we will compare results of the execution with one of the most popular protein database search tools like CD-HIT (Li and Godzik, 2006).

We performed four experiments with different input file sizes and corresponding number of protein sequences. Description of the experiments will be given above each of the following tables which are showing comparison results. CD-HIT denotes existing popular solution, while `pdbs` (**P**rotein **D**ata**b**ase **S**earch tool) denotes the tool used in this thesis.

Comparison overview with CD-HIT

Datasets which have been used in all of the experiments had the structure where we generated *K* sequence seeds and then for each seed we additionally generated similar sequence with 1%, 10%, 25% and 50% of difference with the clustering identity threshold of 0.90 and observing trigrams (k-mer where k is 3). So, for the given *K* seeds, input data size turns out to be *5*K*. Sequence lengths were randomly selected from the interval *[20, 1500]*. We used loose cluster threshold relaxation of 0.80 once when number of loose clusters is greater than the square root of the number of samples that are being processed.

Machine where we run experiment had the following setup as in Table 3.5:

Table 3.5: Server specification

Specification	Info
CPU(s)	32
Thread(s) per core:	2
Core(s) per socket:	8
Socket(s)	2

PDBS run command:

```
time ./pdbs data.in 10 3 10 1 0 100 0.25 0.90 log.out
```

CD-HIT run command:

```
time ./cd-hit -i data.in -o nr100 -c 0.90 -n 3 -M 16000 -d 0
-T 8
```

Table 3.6: Comparison with CD-HIT, K=100, file size=250 KB

Tool	Number of clusters	CPU time	Memory
CD-HIT	301	real=0.206s, user=0.152s	87 MB
PDBS	loose=224, tight=407	real=0.124s, user=0.111s	< 100 MB

Table 3.7: Comparison with CD-HIT, K=1000, file size=2.6MB

Tool	Number of clusters	CPU time	Memory
CD-HIT	3011	real=0.980s, user=4.164s	92 MB
PDBS	loose=425, tight=4133	real=0.368s, user=0.820s	< 100 MB

Table 3.8: Comparison with CD-HIT, K=10 000, file size=26MB

Tool	Number of clusters	CPU time	Memory
CD-HIT	30096	real=15.854s, user=1m38.260	164 MB
PDBS	loose=1645, tight=41211	real=8.190s, user=55.688s	711 MB

Table 3.9: Comparison with CD-HIT, K=100 000, file size=252MB

Tool	Number of clusters	CPU time	Memory
CD-HIT	300877	real=30m18.243s, user=202m8.088s	866 MB
PDBS	loose=5402, tight=412494	real=8m49.086s, user=91m54.068s	1.917 GB

Results discussion

We can see that PDBS is consuming more memory (about 4x more) on the average run, but is also slightly faster, however CD-HIT seems more precise since it provides more exact results which are intuitive for human understanding. E.g. as shown in Table 3.6 we can see that CD-HIT found 300 clusters out of 500 samples, and that was expected since 300 were indeed within 90% of similarity. PDBS found close to 200 loose clusters and close to 400 tight clusters, which means that clustering based on similarity measure which utilizes Jaccard will not in many cases recognize 90% of similarity for the given tight clustering threshold of 0.90. This indicates we need more advanced similarity measure calculation.

4. Technical solution overview

4.1. Design diagram

Solution has been written in C++ programming language, and the code is freely available under MIT licence on `https://github.com/pzuljevic/protein_database_search`.

Having a modular solution makes overall structure more readable and easier to change on the fly while running different experiments. Main modules are as shown in Figure 4.1:

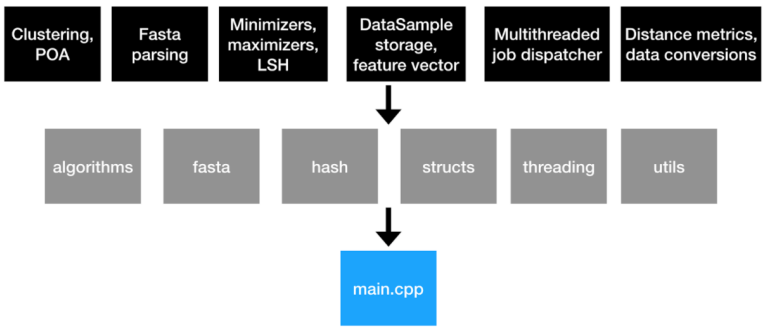

Figure 4.1: The main architectural modules

- **Algorithms**: should contain only algorithmic logic relevant to clustering (e.g. per length, per data content etc.).
- **Fasta**: should contain all of the logic about parsing input files and extracting data which will be used to generated n-dimensional vector of features (integers).
- **Hash**: module which wraps logic to calculate minhash, do LSH and handle minimizers/maximizers.
- **Structs**: data layer which wraps raw data samples and more higher abstraction level data like clusters.
- **Threading**: module relevant to handle multithreading properly, allocate tasks to threads and wait for their execution safely.
- **Utils**: anything that doesn't fit in aforementioned modules, but it has been used across the code.

4.2. Execution phases

Execution is happening through the several phases in form of a chained pipeline as shown in Figure 4.2. First, the input is being processed and feature vectors extracted from the string sequences. Afterwards, protein sequences are being clustered in tight and loosely coupled groups. The final step is to analyze query protein sequences and compare them with the best fitting groups and apply partial order graph alignment.

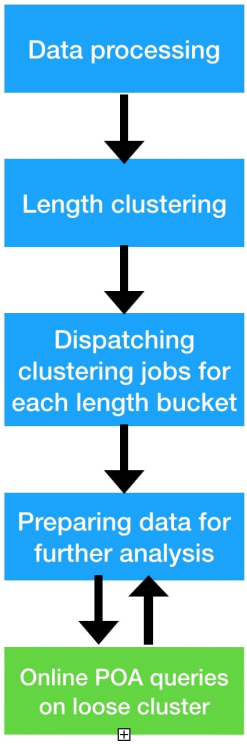

Figure 4.2: The main execution phases

4.2.1. Building database

Primary goal is to build the database and structure the data within it in the form which can be utilized to perform fast queries (i.e. finding highly similar groups).

Data processing

Data processing consists of parsing the input file which is in the form of FASTA sequences. Each FASTA sequence consists of two rows in the file, first row is FASTA header which provides more information about the data sequence itself (e.g. origin, type), and second row contains actual data. Process is reading both information and storing them to the appropriate data structures. It's important to note that processing is happening in multithreaded environment.

Length clustering

Length clustering is providing the ability to isolate input FASTA protein sequences so they could be clustered independently in multithreaded environment. This is relying on the assumption that sequences with the different length for sure are not genetically related and we could discard them from the unique set and analyze in other sets with different threads in parallel.

Dispatching clustering jobs

Loose and tight clustering is the algorithm which operates on the domain of sequences which all have fallen into the same length bucket. Creating loose clusters is dividing the space of search into several areas, while tight clustering is dividing domain of loose clusters to even more granular space. Tight clusters are representing key points for which we know genetical origin and can be used as a strong reference when assigning references to sequences which have never been seen before. Having loose cluster can speed up the search process.

4.2.2. Querying database

Once the database is built and fed with reference sequences, process of running queries and aligning similar sequences can start.

Online POA queries on loose clusters

For each sequence from the list of query sequences, the most similar loose cluster will be specified. The term 'similar' is defined by the measure which has been used in the solution and previously described. Once loose cluster is determined, arbitrary number of 10 tight clusters will be extracted and partial order graph alignment will be applied between query sequence and tight clusters.

4.3. Potential improvements

Few most important areas to improve current solution are related to the loose clustering and prevention of congestion when analyzing input data with significant number of outliers. Possibility that many data samples from input are very different from each other is likely for real datasets. Current solution will generate too many loose/tight clusters since solution is not capable to fit all outliers into one of existing loose clusters. This may cause increase of running time and contribute to slower performance.

The right solution might be on the path to achieve faster detection to determine if new data sample fits certain loose cluster or not. Proposal to address this is based on using Bloom filter. Final algorithm 4 would look like this in that case:

Algorithm 4 Loose clustering (improved)

bf ← Bloom Filter with K hash functions
$W \leftarrow$ set of query protein samples
$L \leftarrow$ set of loose clusters
$lc \leftarrow$ loose clustering threshold
$membership \leftarrow$ uninit
for each: $w \in \mathcal{W}$
    $LBF \leftarrow$ set of loose clusters with a match in Bloom Filter with the query `w`
    $maxS \leftarrow$ the max similarity between query sequence `w` and loose cluster within *LBF*
    $l \leftarrow$ loose cluster with similarity `maxS` with the query `w`
    if $maxS \geq lc$ **then**
        $membership \leftarrow l$
    else
        $l2 \leftarrow$ new loose cluster
        $membership \leftarrow$ `l2`

In the case when many samples do not fit to any cluster (e.g. they are outliers) then we could group them with the rest of outliers to the special cluster so that their presence does not cause too many iterations through the list of loose clusters. Another potential improvement is to define loose cluster quality metric and recalibrate loose clusters (e.g. merge/split them on the fly).

One more improvement could be to utilize partial exact dynamic programming based algorithm to determine the similarity between query sequence and the tight cluster representative (as shown in Table 3.6, Jaccard based similarity measure might impact clustering precision). "Partial" refers to not necessarily execute the algorithm till the end, but just till the point where enough information is obtained to infer if sequence is within the given similarity threshold.

5. Conclusion

Protein database search is challenging problem given that number of newly discovered sequences is increasing and there is no currently fast, sensitive and efficient way to detect similarities between reference query and existing database. In this thesis overview has been given on the approach of using tight and loose clustering to group protein sequences into more and less coupled clusters.

Using loose clusters, enabled us to reduce space search quickly to only the most relevant sequences which we call tight clusters. Tight clusters are represents for the group of very similar sequences and it's expected that query sequence should match more than 95% of amino acid residues of the existing tight cluster represent. POA has been used to get more detailed insight into structural and functional relationship between query sequence and loose cluster representatives via multiple sequence alignment on partially ordered graph.

Results are shown on simulated and real datasets via hierarchical graphs. As a suitable method we established that using (K=3, W=10) minimizers and Jaccard as an distance metric can give correct results quickly. Solution is represented as an C++ binary, and potential improvements for the tool could be on the area of quick detection via Bloom filters to which loose cluster does query sequence fit instead of iterating through existing loose clusters.

BIBLIOGRAPHY

Stephen F Altschul, Warren Gish, Webb Miller, Eugene W Myers, and David J Lipman. Basic local alignment search tool. *Journal of molecular biology*, 215(3):403–410, 1990.

Gelio Alves, Aleksey Y Ogurtsov, and Yi-Kuo Yu. Raid_aps: Ms/ms analysis with multiple scoring functions and spectrum-specific statistics. *PLoS One*, 5(11):e15438, 2010.

Benjamin Buchfink, Chao Xie, and Daniel H Huson. Fast and sensitive protein alignment using diamond. *Nature methods*, 12(1):59, 2014.

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

Moosmosis. Evolution phylogenetic tree interpretation basics, 2018. URL `https://moosmosis.org/2016/07/21/evolution-phylogenetic-tree-interpretation-basics/`.

Christiam National Center for Biotechnology Information. *BLAST (r) Command Line Applications User Manual*. National Center for Biotechnology Information (US), 2008.

Jungkap Park, Paul D Piehowski, Christopher Wilkins, Mowei Zhou, Joshua Mendoza, Grant M Fujimoto, Bryson C Gibbons, Jared B Shaw, Yufeng Shen, Anil K Shukla, et al. Informed-proteomics: open-source software package for top-down proteomics. *Nature methods*, 14(9):909, 2017.

Michael Roberts, Wayne Hayes, Brian R Hunt, Stephen M Mount, and James A Yorke. Reducing storage requirements for biological sequence comparison. *Bioinformatics*, 20 (18):3363–3369, 2004.

Torbjørn Rognes. Faster smith-waterman database searches with inter-sequence simd parallelisation. *BMC bioinformatics*, 12(1):221, 2011.

Joshua Tan, Durga Kuchibhatla, Fernanda L Sirota, Westley A Sherman, Tobias Gattermayer, Chia Yee Kwoh, Frank Eisenhaber, Georg Schneider, and Sebastian Maurer-Stroh.

Tachyon search speeds up retrieval of similar sequences by several orders of magnitude. *Bioinformatics*, 28(12):1645–1646, 2012.

Robert Vaser, Ivan Sović, Niranjan Nagarajan, and Mile Šikić. Fast and accurate de novo genome assembly from long uncorrected reads. *Genome research*, 2017.

Jian Wang, Monika Tucholska, James DR Knight, Jean-Philippe Lambert, Stephen Tate, Brett Larsen, Anne-Claude Gingras, and Nuno Bandeira. Msplit-dia: sensitive peptide identification for data-independent acquisition. *Nature methods*, 12(12):1106, 2015.

Protein Database Search Using Partial Order Alignment

Abstract

Database search is one of the fundamental problems in bioinformatics for which various exact and heuristic methods have been developed. Although execution time for heuristic tools is less several orders of magnitude as compared with well known and used tool BLAST, no tool today is comparable as BLAST regarding sensitivity. The goal of this thesis is to implement the new exact method for local alignment of proteins based on partial order alignment (POA). In ideal case, implemented method should have comparable execution time as BLAST. The main approach for this method is to group proteins from the query with the ones in protein database in order to reduce space of search, i.e. alignment of each query protein with the ones in database. Groups in this sense are partially ordered graphs which contain similar proteins. Once the best graphs with alignments are determined, from those graphs specific alignments will be extracted.

Keywords: Protein, Database search, Partial order alignment graph, Tight, Loose, Clustering, MSA

Pretraživanje baza proteina pomoću djelomično uređenog poravnanja

Sažetak

Pretraživanje proteinskih baza jedan je od temeljnih problema u bioinformatici za koji su razvijene različite egzaktne i heurističke metode. Iako je vrijeme izvođenja heurističnih alata nekoliko reda veličine manje od najpoznatijeg i najkorištenijeg alata BLAST, niti jedan alat ne konkurira alatu BLAST po senzitivnosti. Cilj ovog rada je implementirati novu egzaktnu metodu za lokalno poravnanje proteina temeljenu na djelomično uređenom poravnanju (partial order alignment ili kraće POA). U idealnom slučaju, implementirana metoda bi trebala po brzini izvođenja konkurirati alatu BLAST. Glavna ideja ove metode je grupiranje proteina iz proteinske baze kako bi se smanjio prostor pretraživanja, tj. poravnanje svakog proteina iz upita sa svakim u bazi. Grupe u ovom smislu su djelomično uređeni grafovi koji sadrže slične proteine. Pretraživanje baze se tada svodi na poravnanje proteina iz upita sa svakim grafom u bazi, tj. višestruko poravnanje sekvenci (multiple sequence alignment ili MSA). Kada se odrede grafovi s najboljim poravnanjima, iz samih grafova se izdvoje zasebna poravnanja.

Ključne riječi: Protein, Pretraga baza, Parcijalno uređeni grafovi, Slabo, Čvrsto, Klastering, MSA