Parallelisation of Hierarchical Clustering Algorithms for Metagenomics

Mimi Tantono

June 29, 2015
Preface

I would like to express the deepest gratitude to my thesis supervisor, Torbjørn Rognes, for his continuous support and encouragement during the work on this thesis. Apart from that, I would also like to acknowledge the time and effort that he has spent for implementing the sub-sampling feature in vsearch that has accommodated the experiments on partial datasets in this thesis. And most of all, without his guidance and valuable advice, the completion of this thesis would not have been possible.

In addition, I would also like to thank my close friends and family for all of their love and understanding that has supported me through the entire process.
Abstract

Metagenomics is the investigation of genetic samples directly obtained from the environment. Driven by the rapid development of DNA sequencing technology and continuous reductions in sequencing costs, studies in metagenomics become popular over the past few years with the potential to discover novel knowledge in many fields through analysing the diversity of microbial ecology.

The availability of large-scale datasets increases the challenge in data analysis, especially for hierarchical clustering that has a quadratic time complexity. This thesis presents the design and implementation of a parallelisation method for single-linkage hierarchical clustering for metagenomics data. Using 16 parallel threads, p-swarm was measured to achieve 11 times of speedup. This result shows a significant improvement of execution time while preserving the quality of exact and unsupervised clustering, which makes it possible to hierarchically cluster a larger dataset, for example TARA dataset which consists of nearly 10 million amplicons in just a few hours. Moreover, our method may be extended to a distributed computing model that could further increase the scalability and the capacity to cluster a larger volume of dataset.

Availability: P-swarm was implemented in C++ and source code is available on https://github.com/mimitantono/p-swarm under the GNU Affero GPL.

Keywords: hierarchical clustering; single-linkage; parallel programming; metagenomics.
Contents

List of Figures xi

List of Tables xiii

1 Introduction 1
  1.1 Motivation .................................................. 1
  1.2 Problem Statement ......................................... 2

2 Background 3
  2.1 Metagenomics .................................................. 3
  2.2 Studies Related to Metagenomics ......................... 3
    2.2.1 The Human Microbiome Project (HMP) ............... 4
    2.2.2 The Earth Microbiome Project (EMP) ............... 4
    2.2.3 Example of Applications of Metagenomics on Microbial Ecology ......................... 5
  2.3 Sequencing of DNA .......................................... 5
    2.3.1 History of Influential Sequencing Technology ...... 6
    2.3.2 FASTA Sequence File Format ......................... 8
  2.4 Clustering on Biological Data ............................. 9
    2.4.1 Clustering Algorithms ................................. 10
    2.4.2 Algorithms and Methodologies of Several Heuristic Clustering Tools ...................... 12
    2.4.3 Swarm – a Single-Linkage Hierarchical Clustering Program ................................. 14
    2.4.4 Filtering Based on Q-gram/K-mer Distance .......... 16
  2.5 Parallelisation ............................................. 16
    2.5.1 Levels of Parallelism .................................. 17
    2.5.2 Shared Memory Model .................................. 17
    2.5.3 Distributed Memory Model ...................... 17
    2.5.4 Estimating Speedup Based on Amdahl’s Law and Gustafson’s Law ......................... 18
  2.6 Summary .................................................... 19

3 Design and Algorithms ........................................ 21
  3.1 Challenges and Limitations ................................. 21
  3.2 Concept of Hierarchical Clustering Algorithm .......... 21
  3.3 Parallelisation Strategies ................................. 23
6.2.3 Extending the Algorithm for Other Types of Hierarchical Clustering ............... 74
6.2.4 Storing Pairwise Distances as a Cache ............... 74

7 References ............... 75

8 Appendix: Code Listing ............... 79
  8.1 clusterdata.h ......................... 79
  8.2 clusterdata.cc ......................... 80
  8.3 clusterresult.h ......................... 80
  8.4 clusterresult.cc ......................... 81
  8.5 cluster.h ......................... 83
  8.6 cluster.cc ......................... 84
List of Figures

2.1 Cost of DNA Sequencing from 2001-2014 .......................... 8
2.2 Graph Metrics in Hierarchical Clustering ........................ 11
2.3 Centroid-based clustering based on identity threshold ... 13
2.4 Amdahl’s Law: Speed up of program using multiple 
processors in parallel computing ............................. 19

3.1 Simulation of 3-mer vector comparison of two sequences . 23
3.2 Example of N sequences pairwise matrix ...................... 25
3.3 Example of N sequences pairwise matrix ...................... 26
3.4 Sequence nodes correlated to pairwise matrix in previous 
figure ............................................................. 26
3.5 Economic search on first level connections .................... 28
3.6 Flowchart of the whole process ................................ 31

5.1 Evaluation of execution time on brute-force, first-level, 
multiple-level with various thread numbers for $d = 1$ .... 48
5.2 Evaluation of execution time on brute-force, first-level, 
multiple-level with various thread numbers for $d = 3$ .... 49
5.3 Evaluation of speedup of brute-force, first-level, multiple-
level with various thread numbers for $d = 1$ ............... 50
5.4 Evaluation of speedup of brute-force, first-level, multiple-
level with various thread numbers for $d = 3$ ............... 50
5.5 Evaluation of memory consumptions of brute-force, first-
level, multiple-level with various thread numbers for $d = 1$ 51
5.6 Evaluation of memory consumptions of brute-force, first-
level, multiple-level with various thread numbers for $d = 3$ 51
5.7 Four Level Economic Search .................................. 53
5.8 Empirical Run-time Growth for TARA dataset ......... 57
5.9 Empirical Run-time Growth for EMP dataset ........ 57
5.10 Evaluation of execution time for different values of $d$ on 
BioMarKs dataset .............................................. 59
5.11 Evaluation of memory consumption for different values of 
$d$ on BioMarKs dataset ......................................... 59
5.12 Evaluation of total clusters on BioMarKs and EMP_0.0005 
dataset for different values of $d$ ............................. 60
5.13 Execution time comparison for different values of $d$ on 
EMP_0.0005 dataset ............................................. 61
5.14 Memory comparison for different values of $d$ on EMP_0.0005 dataset . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 61
5.15 CPU usage graph of eight threads on clustering with $d = 1$ and $d = 3$ . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 62
# List of Tables

<table>
<thead>
<tr>
<th>Table No.</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Comparison of Sequencing Instruments</td>
<td>7</td>
</tr>
<tr>
<td>2.2</td>
<td>Different definitions of sequence identity/similarity</td>
<td>14</td>
</tr>
<tr>
<td>4.1</td>
<td>Summary of Test Datasets</td>
<td>43</td>
</tr>
<tr>
<td>5.1</td>
<td>Execution time of brute-force, first-level, multiple-level with various thread numbers</td>
<td>48</td>
</tr>
<tr>
<td>5.2</td>
<td>Speedups of brute-force, first-level, multiple-level with various thread numbers</td>
<td>49</td>
</tr>
<tr>
<td>5.3</td>
<td>Memory consumptions of brute-force, first-level, multiple-level with various thread numbers</td>
<td>51</td>
</tr>
<tr>
<td>5.4</td>
<td>Dataset of different average length</td>
<td>52</td>
</tr>
<tr>
<td>5.5</td>
<td>Execution times of dataset of different average length</td>
<td>54</td>
</tr>
<tr>
<td>5.6</td>
<td>Clustering results of dataset of different average length</td>
<td>54</td>
</tr>
<tr>
<td>5.7</td>
<td>Memory consumptions of dataset of different average length</td>
<td>55</td>
</tr>
<tr>
<td>5.8</td>
<td>Datasets of different size</td>
<td>55</td>
</tr>
<tr>
<td>5.9</td>
<td>Execution times on datasets of different size</td>
<td>56</td>
</tr>
<tr>
<td>5.10</td>
<td>Clustering result of different problem size</td>
<td>58</td>
</tr>
<tr>
<td>5.11</td>
<td>Execution times and memory consumptions for different values of $d$ on BioMarKs dataset</td>
<td>59</td>
</tr>
<tr>
<td>5.12</td>
<td>Execution times and memory consumptions of different values of $d$ on EMP_0.0005 dataset</td>
<td>60</td>
</tr>
<tr>
<td>5.13</td>
<td>Time profile summary for $d = 1$</td>
<td>62</td>
</tr>
<tr>
<td>5.14</td>
<td>Time profile summary for $d = 3$</td>
<td>63</td>
</tr>
<tr>
<td>5.15</td>
<td>Operations count of first-level method on BioMarKs and EMP_0.0005 dataset</td>
<td>64</td>
</tr>
<tr>
<td>5.16</td>
<td>Operations count of multiple-level method on BioMarKs and EMP_0.0005 dataset</td>
<td>64</td>
</tr>
<tr>
<td>5.17</td>
<td>Execution Time (Seconds) Comparison with Other Clustering Tools</td>
<td>67</td>
</tr>
<tr>
<td>5.18</td>
<td>Memory (MB) Comparison with Other Clustering Tools</td>
<td>68</td>
</tr>
<tr>
<td>5.19</td>
<td>Clusters Count Comparison with Other Clustering Tools</td>
<td>69</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

1.1 Motivation

The approximated amount of prokaryotes and their cellular carbon on earth is $4 - 6 \times 10^{30}$ cells, most occurs in the open ocean ($1.2 \times 10^{29}$), in soil ($2.6 \times 10^{29}$), in oceanic ($3.5 \times 10^{30}$), and terrestrial subsurfaces ($0.25 - 2.5 \times 10^{30}$) [Whitman, Coleman, & Wiebe, 1998]. Although the number of prokaryotes on animals (including human) was found not to constitute a great proportion of prokaryotes, they play an important role in nutrition and disease. For instance the human body is composed of approximately $10^{13}$ eukaryotic cells, while the body surfaces and gastrointestinal canals, i.e. oesophagus, stomach, small intestines, cecum, and large intestines - of humans may be colonised by as many as $10^{14}$ indigenous prokaryotic and eukaryotic microbial cells [Savage, 1977]. These residents microbiome and human genome somehow form a mutualistic symbiosis relationship to a certain extent that they are dependent with each other where the disruption of one may affect the well-being of the others, for example, microbiome provide enzymes for digestion, and overgrowth of intestinal flora may cause irritable bowel syndrome [Nelson, 2011].

Over more than 99% of microorganisms within all microbial groups (i.e. bacteria, archaea, fungi, viruses, algae and protozoa) are “unculturable” with the current laboratory techniques. For the ones that can be grown in the lab’s artificial environments, the result would unlikely reflect its original gene expression, protein and metabolite profiles in nature. Therefore, the approach to look at microorganisms at their native habitats is crucial for understanding their functions and characteristics. With the development of high-throughput sequencing (HTS) technologies, an enormous number of sequences data is now available without having to go through the process of cultivation. It enables the study of metagenomics, it opens many potential discoveries related to natural microbial diversity and its correlation with stability in natural environments, and it also shifts the weight of analysis to the area of data analysis.

One of the main steps in analysing these redundant biological data
is to cluster similar OTUs (Operational Taxonomy Units) that are close to each other but far from the others, so that further expensive analyses can be performed on the set of grouped data. Hierarchical clustering offers an unsupervised and non-parametric approach to this step but is subject to more challenges due to its complexity in time and space in addition to the high-cost pairwise alignment. To anticipate the increasing number of sequences that are now available to be processed, there is a need to work out an approach to attaining scalability with parallelisation while preserving the clustering accuracy of the hierarchical clustering algorithm.

1.2 Problem Statement

Based on the motivations mentioned above, the objective of this study is as following.

- To review the strategies for parallelisation of single-linkage clustering in the context of metagenomics.
- To design and implement such an efficient strategy.
- To evaluate and compare it to existing tools.
Chapter 2

Background

2.1 Metagenomics

Metagenomics was first introduced when in 1998 Handelsmann et al. explored the concept of studying soil microbes by directly accessing its genomes and bypassing laboratories culture [Gilbert & Dupont, 2011]. Since the samples are studied directly without having to rely on microbiological culture, it encourages the studies of organisms that are not easily cultured in the laboratory in addition to saving time and cost for collecting samples. As mentioned before, 99% of all microorganisms are not culturable while according to Eckburg et al. only 20% of human intestinal microbiota are culturable (as cited in [Nelson, 2011]).

A pilot project of environmental study was conducted in 2004 by J. Craig Venter Institute in the Sargasso Sea – named before the Sargassum brown seaweed genus that is floating on the surface – which is located in Atlantic Ocean near Bermuda [Sleator, Shortall, & Hill, 2008]. Using whole-genome shotgun sequencing, the outcome of this “largest-ever” metagenomics project at the time was the identification of “1,800 new species and more than 1.2 million new genes” [Larkman, 2007]. Apart from its application on the natural environment, metagenomics can also be performed on several locations of the human body, e.g. skin, mouth, intestine, with the goal to potentially aid improving human health.

2.2 Studies Related to Metagenomics

Microbial ecology focuses on two areas of study: (i) microbial diversity, (e.g. phylogenetic diversity, species diversity, genotype diversity, gene diversity, evolitional diversity, metabolic diversity, functional diversity) and (ii) microbial activity [Xu, 2006]. The characteristics of metagenomics that does not require DNA culture has driven more studies about microbial diversity that are rich with promises in bringing useful applications on the environment and human life. There are two initiatives related to metagenomics study which had eventually expanded to worldwide where researchers from all around the world
collaborate by making their data and finding available to be freely used.

2.2.1 The Human Microbiome Project (HMP)

In the long run the study of human microbiome is expected to provide potential interventions on altering the state of one’s microbiome compositions using antibiotics, prebiotics (short-chain carbohydrates), probiotics (microorganisms such as bacteria or yeast), or synbiotics (combinations of the former three components) to enhance human health or even in curing and preventing diseases [Nelson, 2011]. Furthermore, the evidence indicates that either pathogen or beneficial microorganisms evolve as a human does, hence to understand human microbiome is important and it will give much interest into the medicine related fields of study.

With the reasons mentioned above, the interest in studying the microbiome increased with the advent of microbial communities metagenomics analyses and the ability to produce the whole genome sequence of bacteria. In 2008 this became a reality when HMP was selected as an initiative in the National Institute of Health (NIH) Roadmap for Biomedical Research with a total investment over $150 million with the aims to produce reference genome sequences for microbes, determine their structures in the 18 body sites of healthy subjects, develop new laboratory and computational approaches, perform projects investigating human diseases, and investigate the ethical, legal and social impact of these research [Weinstock, 2011].

2.2.2 The Earth Microbiome Project (EMP)

The concept of EMP was born as the main outcome of The Terabase Metagenomics Workshop in summer 2010, and its name was given to show respect to HMP as the pioneer in the field. The workshop which was sponsored by the Institute for Computing in Science (ICiS) aimed to address the challenge of finding the best possible way to utilise the discovery of next-generation sequencing platforms such as the Illumina which can sequence 250 billion DNA base pairs in as short as 8 days [Gilbert et al., 2010]. Further analysis such as metagenomics, meta-transcriptomics and amplicon sequencing will be executed on the collected dataset to assemble a “global Gene Atlas” which consists of the information of proteins, environmental metabolic model, and a portal for data visualisation.

Following the workshop, the EMP was launched consequently in August 2010 with the objectives mentioned above. Through “crowdsourcing, soliciting donations of samples” from more than 200 researchers around the world within numerous different microbial ecology disciplines – human animal, plan, terrestrial, marine, freshwater, sediment, air, built-environment, and every intersection of these ecosystems – in approximately 4 years (as of July 2014) EMP had acquired and processed over 30,000 environmental samples consist of more than 40 dif-
ifferent ecologies of the earth, generated 16S rRNA amplicon data and releasing them to QIIME (Quantitative Insights into Microbial Ecology) database [Gilbert, Jansson, & Knight, 2014]. EMP’s mission is to continue to grow and adapt, exploring metagenomics analysis, and adding new avenues including potentially extra-terrestrial locations.

2.2.3 Example of Applications of Metagenomics on Microbial Ecology

We collected some examples of potential uses of metagenomics on microbial ecology as following:

2.2.3.1 Application on Plant-Microbe Interactions

Soils are the most genetically rich environments on earth and because of that its metagenomics analysis was fairly neglected due to the technical difficulties of DNA analyses. Plant and soil microbes have a strong relationship with each other, therefore studying soil microbes will lead to the improvement of planting and cultivation techniques, immunisation and fertilisation resulting in healthy and productive crops (as cited in [Kuiper, Lagendijk, Bloemberg, & Lugtenberg, 2004]).

2.2.3.2 Application on Bioremediation

The concept of bioremediation is to clean up the polluted environment with as little human intervention as possible so that the advantages of cost saving and permanent effect can be achieved [George, Stehnuit, Agathos, & Marco, 2010]. Some bacteria and funguses have the metabolic characteristics which chemically purify hazardous pollutants and metagenomics is a convenient tool to understand these physico-chemical characteristics.

2.2.3.3 Application on Industrial Bioproducts

Microbial genomics were already used before to develop enzymes that are useful in biotechnology such as: antibiotics and secondary metabolites, organonitriles and industrial synthesis (synthesis of plastics, fibre, fumigant, dyestuffs, etc.), food processing (e.g. modification of lipids in the dairy products manufacturing), and many more [Wong, 2010]. There are still a large number of unknown enzymes that yet to be discovered and will contribute to boosting the efficiency of industrial processes.

2.3 Sequencing of DNA

DNA sequencing is the process of identifying the occurrences of four different types of nucleotides namely adenine, guanine, cytosine, and thymine within a DNA strand of the cell of an organism.
2.3.1 History of Influential Sequencing Technology

In the early days, the first sequencing technology was developed by Frederick Sanger in 1977, referred as "Sanger" sequencing or Chain-terminator sequencing. This sequencing technology was considered to be easy to use and reliable to determine nucleotide sequences in a single stranded DNA, therefore it was widely used for around 25 years. However, Sanger technology is an expensive and low-throughput technology aside from the limitation of it being biologically biased [Bragg & Tyson, 2014]. In order for the method to give a reliable result, one must satisfy various criteria related to compatibility of the samples to be sequenced with *Escherichia Coli* as the DNA template. Despite these reasons, this technique is still useful on smaller-scale projects that require long contiguous DNA sequence reads.

Pyrosequencing was invented 20 years later, and precisely in 2005 Roche 454 successfully developed a commercial DNA sequencer with the performance to generate about 500 million bases of raw sequence in just a few hours (as cited in [Pettersson, Lundeberg, & Ahmadian, 2009]). A few years later during 2008, Illumina and Applied Biosystems SOLiD (Sequencing by Oligo Ligation and Detection) have introduced the sequencing systems that offer even higher throughput at billions of bases in a single run. To obtain such the capability of generating such a generous number of base pairs, both of the novel methods rely on the parallelisation of spatially separated clonal amplicons that leads into a much higher throughput.

The comparison of characteristics and details of each sequencing technology are shown on table 2.1 [Glenn, 2011; Liu et al., 2012]. The cost, accuracy, read length, runtime and output of each technology were retrieved from "2014 NGS Field Guide: Overview"\(^1\). The numbers represented in the table were taken from the following instruments that were selected as the example from each platform:

- 454: 454 FLX+
- Illumina: Illumina HiSeq 2500 - high output v4
- Ion Torrent: Ion Torrent - PGM 318 chip

Sanger and PacBio each has only one model: Applied Biosystems 3730 (capillary) and Pacific Biosciences RS II.

---

\(^1\)http://www.molecularecologist.com/next-gen-fieldguide-2014/
<table>
<thead>
<tr>
<th>Platform</th>
<th>Sanger</th>
<th>454</th>
<th>Illumina</th>
<th>Ion Torrent</th>
<th>PacBio</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Current Company</strong></td>
<td>Applied Biosystems</td>
<td>Roche</td>
<td>Illumina</td>
<td>Life Technologies</td>
<td>Pacific Bioscience</td>
</tr>
<tr>
<td><strong>Former Company</strong></td>
<td>Life Technology</td>
<td>454</td>
<td>Solexa</td>
<td>Ion Torrent</td>
<td>N/A</td>
</tr>
<tr>
<td><strong>Sequencing Method</strong></td>
<td>Chain-termination</td>
<td>Synthesis (Pyrosequencing)</td>
<td>Synthesis</td>
<td>Synthesis (H+ detection)</td>
<td>Synthesis</td>
</tr>
<tr>
<td><strong>Claim to Fame</strong></td>
<td>High quality; Long read length</td>
<td>First NGS; Long reads</td>
<td>First short-read sequencer; Current leader in advantages</td>
<td>First Post-light sequencer; First system &lt; $100,000</td>
<td>First real-time single-molecule sequencing</td>
</tr>
<tr>
<td><strong>Cost / million bp</strong></td>
<td>$2,307</td>
<td>$9.5</td>
<td>$0.06, $0.03</td>
<td>$0.79, $0.46</td>
<td>$1.11</td>
</tr>
<tr>
<td><strong>Final error rate</strong></td>
<td>0.1-1%</td>
<td>1%</td>
<td>~ 0.1%</td>
<td>~ 1%</td>
<td>≤ 1%</td>
</tr>
<tr>
<td><strong>Read Length</strong></td>
<td>650 bp</td>
<td>650bp</td>
<td>50 bp / 250 bp</td>
<td>200 bp / 400 bp</td>
<td>3,000 bp</td>
</tr>
<tr>
<td><strong>Time / Run</strong></td>
<td>2 hours</td>
<td>20 hours</td>
<td>40 hours / 6 days</td>
<td>4.4 hours / 7.3 hours</td>
<td>2 hours</td>
</tr>
<tr>
<td><strong>Output / Run</strong></td>
<td>62.4 Kbp</td>
<td>650 Mbp</td>
<td>100 Gbp / 500 Gbp</td>
<td>950 Mbp / 1.9 Gbp</td>
<td>90 Mbp</td>
</tr>
</tbody>
</table>

Table 2.1: Comparison of Sequencing Instruments
The discovery of HTS (High-Throughput Sequencing) has opened the window of more scientific research, expanding our knowledge into the area that was not available before due to the limitation of sequencing technology. The progressive development of sequencing methods not only allowed researchers to directly sequence without having to culture the environmental or human body samples, it also substantially reduces the cost of DNA sequencing. Figure 2.1 shows that the cost of sequencing in the past 10 years has dropped more than predicted by Moore’s law (computing power would double every two years).

2.3.2 FASTA Sequence File Format

There are many file formats exists in the field of Bioinformatics. One of them that is the oldest and simplest is FASTA sequence format [BioPerl, 2014], which is a text file containing one or more DNA or protein sequences. Each sequence is represented by two lines; the first line as the header must be marked by character > followed by a description, and the second line contains symbols of amino acids or nucleotides, usually in capital.

There is no formal standard of what a header line should contain, and which character to be used as the separator. Most projects usually have their own way to specify some information in the header line. For example an "NCBI" (National Center for Biotechnology Information) formatted FASTA sequence header usually includes the
database identifier number, database code, accession number (identifier number for a sequence), and Locus name, delimited by a pipe character, e.g.:

>gi|142864|gb|M10040.1|BACDNAE B. subtilis dnaE gene encoding DNA primase, complete cds

Most of the heuristic clustering tools such as CD-HIT, USEARCH, VSEARCH, DNAACLUST, and so on, recognise abundance by the label "size", which is placed after the description of the sequence and delimited by a semicolon character, e.g.:

>0b65e14109c71683d27299a3fc5fe362fca9f18d; size = 2034;

There is also no formal standard file extensions for a FASTA sequence file. Usually *.fas* or *.fasta* is used for any generic fasta file, *.fna* for any fasta containing nucleic acids, *.ffn* for coding regions of a genome, *.faa* for amino acids, and *.frn* for non-coding RNA regions.

### 2.4 Clustering on Biological Data

Given a set $S$ of $n$ vertices in $\mathbb{R}^d$, a clustering problem is defined as the operation of classifying $S$ into $k$ clusters – usually unsupervised – such that the points in each cluster are either close to each other within the distance of $d$ or close to some cluster center or centroid within the distance of $d/2$. In the context of data-mining where there exists an abundant number of raw data, clustering is regularly used for compression, reducing redundancy, and grouping of data so that a more expensive analysis can later be performed on the smaller amount of data.

Unsupervised clustering is one of the approaches that are used in metagenomics to assign DNA sequences into operational taxonomy units (OTUs) or phylotypes — a unit of organism in microbial diversity — in order to estimate the richness and diversity of a microbiome community. OTUs in metagenomics are commonly specified based on the genetic distances between sequences since DNA sequences are the only available data for these organisms.

Clustering is relatively more complex when performed on biological data because a sequence database is a high dimensional data that exists in non-metric spaces. Given sequences $x, y,$ and $z$ and a scoring function $d$ to measure the similarity between one sequence with another, the triangle inequality $d(x, y) \leq d(x, z) + d(z, y)$ does not strictly hold [Cai & Sun, 2011]. However, a Monte Carlo experiment was performed by Cai et al. to measure the significance of the violation and it was found out that there were only 7 out of 100K trials that did not comply with the inequality.

On the other hand, HTS uses PCR (Polymerase Chain Reaction) to amplify DNA samples, and this technique is prone to produce a chimeric sequence. As cited by Edgar et al., up to 46% of 16S database is composed of chimeric sequences, mainly chimeras.
with two segments (bimeras) and sometimes chimeras with more than two segments (multimeras) [Edgar, Haas, Clemente, Quince, & Knight, 2011]. Existing studies indicate that without preprocessing, clustering may give an overestimation of OTUs while data corrected with preprocessing would significantly improve the quality of the clustering [May, Abeln, Crielaard, Heringa, & Brandt, 2014; Bonder, Abeln, Zaura, & Brandt, 2012]. This shows as well that articulation processes like preprocessing prior or postprocessing succeeding to the clustering itself must not be neglected. QIIME is a popular open-source bioinformatics pipeline on which the entire workflow of data analysis – from raw sequencing data to demultiplexing and quality filtering, OTU picking, taxonomic assignment, phylogenetic reconstruction, diversity analyses and visualisations – may be performed [Caporaso et al., 2010].

2.4.1 Clustering Algorithms

2.4.1.1 K-means Clustering

Clustering is a classic problem. In 1967 MacQueen presented a simple unsupervised algorithm called k-means clustering which can be extended to be applied on several problem domains beyond the Euclidean distance [MacQueen et al., 1967]. The idea (and also the drawback) of this algorithm is to assume that there exist $k$ centroids – which could be defined randomly – and then iterate each vertex to measure the distance between itself and the available centroids. Later the vertex will be assigned to the nearest cluster – having the minimum distance with the centroid, and the centre of cluster shall be re-calculated accordingly since its members have changed [Polanski, 2007].

2.4.1.2 Hierarchical Clustering

Another solution to the clustering problem is the hierarchical clustering which constructs a minimum spanning tree called dendrogram by first calculating a symmetrical matrix containing the value of pairwise distances between each vertex with all of the other vertices. To obtain clusters, the tree is cut on some levels, i.e. on the edges with the maximum length decided based on some threshold of the clustering, and hence every cluster is, in fact, some subtree of the complete dendrogram [Polanski, 2007; Olson, 1995]. The maximum length which is also the distance between two clusters can be measured either in graph metrics, where the distances are defined by the cost function of the edges between vertices in the two clusters, or by geometric metrics where the distances are calculated between centroids. A graph metric is generally used in most of the existing clustering programs since it reflects well the characteristic of a biological dataset. The commonly used graph metrics are as following (illustrated on figure 2.2).
**single linkage:** minimum distance between vertices in the two clusters.

\[ D(M, N) = \min_{m \in M, n \in N} d(m, n) \]  

(2.1)

**average linkage:** average distance between vertices in the two clusters.

\[ D(M, N) = \frac{1}{|M||N|} \sum_{m \in M, n \in N} d(m, n) \]  

(2.2)

**complete linkage:** maximum distance between vertices in the two clusters.

\[ D(M, N) = \max_{m \in M, n \in N} d(m, n) \]  

(2.3)

There have been developed quite a handful number of existing traditional hierarchical clustering programs for biological data, using the most common strategies – grouping of 16S rRNA sequences into some operational taxonomic unit (OTU) – most of them having the complexity of at least \(O(N^2)\) where \(N\) is the number of sequences, for example DOTUR in 2005 [Schloss & Handelsman, 2005] which was later integrated into Mothur in 2009 [Schloss et al., 2009], and ESPRIT in 2009 [Sun et al., 2009] succeeded by its improved version ESPRIT-tree [Cai & Sun, 2011], and SLP [Huse, Welch, Morrison, & Sogin,
ESPRIT was developed to reduce the complexity and memory requirement of Mothur, and ESPRIT-tree is an improved version of ESPRIT which achieves the same level of accuracy with its predecessor but having a quasi-linear computational complexity [Chen, Zhang, Cheng, Zhang, & Zhao, 2013] through employing a \textit{k-mer filtering} scheme.

The time and space complexity which grows quadratically with the problem size becomes the main aspect that makes hierarchical clustering a challenging problem. Parallelising a hierarchical clustering algorithm is also a difficult problem as the general algorithm comprises a sequential workflow and every step requires access to the entire dataset. For this reason, heuristic clustering programs are currently more widely-used for clustering large amount of data.

\subsection{2.4.1.3 Heuristic Clustering}

A heuristic approach can be generally defined as the experience-based approach to problem solving through self-learning or trial and error so that it can produce results in a reasonable amount of time (in contrast to the classical approach). Usually, this method does not give an optimum solution, but just a local optimum or an estimation to the global optimum which is good enough for some applications.

Traditional heuristic clustering methods for biological data usually involve building a guide tree which is later used to construct the multiple sequence alignment (MSA) of every sequence [Ghodsi, Liu, & Pop, 2011]. Some of these methods select a sequence as a seed for the first cluster and then sequentially adding next sequences to this cluster if they are close, or form a new cluster if otherwise [Ghodsi et al., 2011; Chen et al., 2013].

There have been many heuristic clustering programs that were developed for biological data (in the order of the time published) such as: CD-HIT [Li & Godzik, 2006], UCLUST [Edgar, 2010], GramCluster [Russell, Way, Benson, & Sayood, 2010], DNACLUST [Ghodsi et al., 2011], CD-HIT (V4.6) [Fu, Niu, Zhu, Wu, & Li, 2012], sumaclust [Mercier, Boyer, Bonin, & Coissac, 2013] and VSEARCH [Flouri et al., 2015].

\subsection{2.4.2 Algorithms and Methodologies of Several Heuristic Clustering Tools}

All the heuristic tools CD-HIT, USEARCH, VSEARCH, DNACLUST, and sumaclust use a greedy incremental clustering algorithm as illustrated on figure 2.3. The clustering starts by assigning the first sequence in the input as the seed of the first cluster. Then, the seed will be compared with the remaining sequences. Every sequence that has a similarity above the required threshold will become a new group member of the cluster, otherwise it will be assigned as the seed of the next cluster. The same process happens again and again until every
sequence in the input belongs to a cluster. In this clustering method, the radius of the cluster is fixed to the identity threshold and the diameter of the cluster cannot be more than twice of the cluster’s radius.

Figure 2.3: Centroid-based clustering based on identity threshold

Although all tools employ the same greedy clustering algorithm, there are small differences between each tool in terms of similarity definition, sorting of input, heuristic techniques to increase the speed or precision, and parallelisation mechanism. Almost all of the tools are highly customisable, and they are also equipped with a set of default options that become the outline of their special characteristics.

Each tool uses a similar heuristic method related to k-mer filtering to avoid expensive pairwise alignments. CD-HIT uses "short-word counting and indexing table" to rule out unnecessary alignments. In its shared-memory parallelised version written with OpenMP, one thread is assigned to handle the writing and reading to/from a global indexing table while the remaining threads are performing the clustering procedure [Fu et al., 2012]. USEARCH and VSEARCH use a similar filtering strategy by counting the number of common unique words or k-mers. Sumaclust is implemented with a "lossless k-mer filter" followed by a banded alignment algorithm that is very efficient for a high threshold, both filter and alignment are parallelised with SIMD [Mercier et al., 2013].

USEARCH compares the seed with the targets in a decreasing order by total common unique words between the seed and the target, the sorted list is referred as the "U vector". A search is terminated after some defined number of accepts (by default 1) and rejects (by default 8) because the total common unique words correlate well with similarity [Edgar, 2010]. Which means that cluster members are more likely to be found in the first few targets that share the most common unique words. Although this kind of arrangement could increase the speed, but
sequences which fall beyond the assumption might be missed out.

Earlier versions of UCLUST employed CD-HIT definition as the similarity definition, but it uses BLAST identity as the default option since version 6. DNAACLUST uses the (semi-)global alignment score while sumaclust by default normalises identity score by alignment length. The identity definition in VSEARCH is customisable to up to five different definitions while the default definition is based on alignment length excluding gaps. Each definition were summarised as table 2.2.

<table>
<thead>
<tr>
<th>Definition</th>
<th>Formula</th>
</tr>
</thead>
<tbody>
<tr>
<td>CD-HIT definition</td>
<td>1 - ( \frac{\text{match}}{\text{shorter_sequence_length}} )</td>
</tr>
<tr>
<td>BLAST identity</td>
<td>1 - ( \frac{\text{match}}{\text{alignment_length} + \text{gaps}} )</td>
</tr>
<tr>
<td>(semi-)global alignment score</td>
<td>( \frac{\text{edit_distance}}{\text{shorter_sequence_length}} )</td>
</tr>
<tr>
<td>VSEARCH identity</td>
<td>( \frac{\text{match}}{\text{alignment_length} - \text{gaps}} )</td>
</tr>
</tbody>
</table>

Table 2.2: Different definitions of sequence identity/similarity

By default, UCLUST does not sort input, but it is customisable to sort by length or abundance. CD-HIT and DNAACLUST sort input by length, while sumaclust sorts input by abundance because “true sequences should be more abundant” [Mercier et al., 2013]. VSEARCH support both sorting types and the user must specify which one to use.

By default sumaclust clusters the input with the "exact" option. The "exact" definition here means that a sequence will be assigned to the cluster of which seed has the most similarity with the sequence, instead of assigning it to the first cluster that was found with the similarity above threshold. DNAACLUST flags unclustered sequences within twice the radius from the seed so that they would not be picked up as a cluster centre, but may be included in a cluster with an un-flagged cluster centre [Ghodsi et al., 2011]. This is to prevent the forming of some overlapping clusters with centroids less than twice of the cluster radius.

### 2.4.3 Swarm – a Single-Linkage Hierarchical Clustering Program

Swarm is an exact, agglomerative, unsupervised, and single-linkage clustering method that produces meaningful OTUs and it has less dependency to the clustering parameters [Mahé, Rognes, Quince, de Vargas, & Dunthorn, 2014]. Swarm was developed to address two major problems that usually happen in the greedy clustering methods, which are:

- Each organism evolved with a different speed while greedy clustering methods use a fixed global threshold that cannot fit both
slow-evolving and rapidly-evolving lineages at the same time.

- As greedy clustering methods are operating according to the input order of amplicons, previous centroids were not re-evaluated as the clustering proceeds which could generate an inaccurate result.

In a comparison study A. May et al. ran 11 different clustering algorithms on some mock dataset which were simulated to mimic raw experimental pyrosequencing data [May et al., 2014]. The outcome of the study showed that despite providing the quality of minimum underestimation when performed on some chimera checked and de-noised data, swarm has a significantly shorter running time compared to other algorithms of a similar quality.

Single linkage is prone to noise data such as a set of vertices forming a long chain connecting supposedly two different clusters, will be identified as one cluster [Jain, Murty, & Flynn, 1999]. This problem is handled in Swarm with a companion algorithm that could identify probable amplicon chains and break them into independent OTUs.

Single-linkage is the only metric that satisfy both SANN (same agglomerative nearest neighbour) property and reducibility property [Olson, 1995]. Suppose there exist cluster i and j to be agglomerated into cluster k, SANN means that nearest neighbours of k is the nearest neighbours that i and j used to have, while reducibility means that k will never be closer to any clusters that i and j were not. Taking advantage of these properties makes it possible to incrementally cluster one item at a time, which gives the benefit of less memory requirement for storing distance values, and less running time [Jain et al., 1999].

Following this property, swarm has the workflow of agglomerating one cluster at a time, and the process is repeated until every amplicon belongs to a cluster. The clustering starts by assigning the first amplicon in the pool as the seed for the first cluster. The seed is then compared against all of the amplicons that remain in the pool. Amplicons that have a similarity above the threshold will be assigned as the subseeds of the cluster and removed from the pool. To get a second layer of subseeds, each subseed will be compared again but only with the amplicons that have no more than 2d differences with the seed. This filtering step is based on the triangle inequality that the distance between amplicons that have more than 2d differences with the seed cannot have d or fewer differences with the subseed, since the subseed has no more than d difference with the seed. This can be expanded to amplicons that have more than (k + 1) differences with the seed cannot have d differences with a k-seed. This process is repeated until no more subseeds can be assigned to the cluster. Then the next amplicon will be assigned as the seed for the second cluster, and the same iterations will be performed until the pool is empty.
2.4.4 Filtering Based on Q-gram/K-mer Distance

Computing similarity between sequences is the core of any clustering algorithm. A sequence belongs to a cluster when the similarity between the sequence and the cluster is above a defined threshold while the definition of similarity may be different between one algorithm and another. For example in the heuristic algorithm a sequence is usually compared to the centroid of the cluster while in hierarchical clustering a sequence is compared to all members of the cluster.

Filtering based on k-mer distance is commonly used in many existing clustering tools seeing that a sequence alignment is expensive in both time and memory. Dynamic programming methods like Needleman-Wunsch algorithm for global alignments and Smith-Waterman algorithm for local alignments both have the time and space complexity of $\mathcal{O}(MN)$ when using the affine gap penalty.

Following definitions from [Ukkonen, 1992], a q-gram or k-mer can be defined as any string $s = a_1 a_2 ... a_k$, $s \in W^q$ where $W^q$ denotes the set of all permutations of the set $W = \{'A', 'C', 'T', 'G'\}$, the only possible nucleotides in a DNA sequence. Since $|W| = 4$, it follows that $|W^q| = 4^q$.

Let $x = a_1 a_2 ... a_n$ be a string in $W^*$, if $a_ia_{i+1}a_{i+q-1} = v$ for some $i$, then $x$ has the occurrence of $v$ in $x$. Let $G(x)[v]$ be the total number of the occurrences of $v$ in $x$. The q-gram profile of $x$ is the vector $G_q(x) = (G(x)[v])$ for all $v \in W^k$. Let $x, y$ be strings in $W^*$, the q-gram distance between $x$ and $y$ is

$$D_q(x, y) = \sum_{v \in W^q} |G(x)[v] - G(y)[v]|$$

Furthermore, the q-gram distance $D_q(x, y)$ can be evaluated in time $\mathcal{O}(|x| + |y|)$ and in space $\mathcal{O}(|W|^q + |x| + |y|)$. The q-gram distance can be used to predict the edit distance, based on the fact that one edit operation can alter at most $q$ number of q-grams.

2.5 Parallelisation

Parallelisation can be defined as an effort to organise a set of independent subtasks to be executed simultaneously so that the running time of an algorithm can be reduced in a significant way.

There have been several attempts before on designing parallel algorithms for hierarchical clustering algorithms, starting from a parallel algorithm on PRAM proposed by Olson [Olson, 1995], followed by a parallel version of POP (Partially Overlapping Partitioning) algorithm by Dash et al. [Dash, Petrutiu, & Scheuermann, 2004], a parallel algorithm involving MST construction [Olman, Mao, Wu, & Xu, 2009], PARABLE (Parallel Random-Partition Based Hierarchical Clustering) with MapReduce framework [Wang & Dutta, 2011], and other studies that were related to hardware-based approach such as SIMD and optical bus.
Following we highlight several important concepts related to parallel computing [Rauber & Rünger, 2013; Barney et al., 2010].

2.5.1 Levels of Parallelism

There are three levels of parallelism in parallel computing: bit-level parallelism, instruction-level parallelism, and task parallelism. Bit-level parallelism is achieved by increasing processor word size so that single operation will be able to cover a greater size of variables. For example, a 64-bit processor would need only one operation to sum up two 62-bit integers while a 32-bit processor would need two operations.

Instruction-level parallelism means to improve the execution time by increasing number of instructions that can be fetched and executed simultaneously by the processor. Instruction-level parallelism also depends on compiler design and family of the processor where nowadays most modern processors support multiple stages of instruction pipelines.

Task-level parallelism is the parallelism on the programming level – threads programming – where tasks are designed in a way that they can be executed concurrently by multiple processors.

2.5.2 Shared Memory Model

A shared memory model is a memory organisation such that processes share information with each other from a main memory. Examples of shared memory computing model are multithreading and OpenMP for multiple nodes. This model is suitable for programs that require or could benefit from sharing information among its thread. When processes share memory, phenomena like dirty read, phantom read, unrepeatable read, or undefined behaviour might happen, therefore the concurrency must be handled through the synchronisation among threads. There are four different isolation levels that can be implemented when accessing memory, depending on the level of exclusivity that need to be achieved – from lowest to highest:

- concurrent read concurrent write (CRCW)
- concurrent read exclusive write (CREW)
- exclusive read concurrent write (ERCW)
- exclusive read exclusive write (EREW)

2.5.3 Distributed Memory Model

Distributed memory model is the memory organisation such that processes run independently with their own memory and do not share any information with each other. This model is suitable for programs that do not require nor benefit from sharing information among its processes.
A parallel distributed programming model like MapReduce is mainly composed of two steps: \textit{map} and \textit{reduce}. A problem is first partitioned into multiple \textit{maps} that will be solved individually and in parallel on each of the worker nodes. The output of each \textit{map} will then be written on a temporary storage. These outputs will be sorted or shuffled and grouped into a set of \textit{reduce} operations that will be processed by the worker nodes to conclude a final result of the problem.

2.5.4 Estimating Speedup Based on Amdahl’s Law and Gustafson’s Law

Speedup is a way to measure how much the code improves in running time after being parallelised. It is generally defined as \( \text{Speedup} = \frac{T_s}{T_p} \) where \( T_s \) is time for the sequential program and \( T_p \) is time for the parallel version while \( p \) is the number of threads. There are two popular laws which estimate the theoretical maximum speedup that one can obtain from parallelising a computation. Amdahl’s law observes the speedup based on a fixed problem size, but varying number of processors and the parallel/serial fraction of a parallel program, while Gustafson’s law also includes the growing of the problem size into the estimation.

In 1967, Gene Amdahl proposed on Amdahl’s law which suggests the expected speedup of a parallel program as:

\[
\text{Speedup} = \frac{s + p}{s + \frac{p}{N}} = \frac{1}{s + \frac{p}{N}} \tag{2.5}
\]

where \( s \) is the portion of the program that cannot be made parallel, \( p \) is the portion of the program that can be made parallel and \( N \) is the number of processors or threads.

This means that there is a maximum speedup where adding more processors no longer gives more speedup, while the portion of code that can be ‘greatly affects how much of maximum speedup [Amdahl, 1967]. Figure 2.4 illustrates the calculated expected speedup as the definition above as more processors were added into the parallel program.

This proposal was revisited in 1988 by John Gustafson who proposed a remark to the Amdahl’s law stating that as problem size grows, the speed up of a parallel program would grow linearly [Gustafson, Montry, & Benner, 1988]. Gustafson’s law is applicable if the growing of problem size and processor number do not increase the complexity of the serial portion so that the parallel portion will have a runtime of \( 1/N \) of the serial runtime. Speedup is defined as:

\[
\text{Speedup} = N - s(N - 1) \tag{2.6}
\]

where \( N \) is the number of processors and \( s \) is the serial portion of a program.
2.6 Summary

Metagenomics has become more popular as the result of the revolution in sequencing technology. DNA database which used to be built upon lab cultivation and human curation now is being replaced by HTS to a limited extent. As a result of the revolution more computing is required to process these data and to eliminate redundant data through clustering so that more costly analysis can be performed on the smaller dataset. Clustering of a sequence dataset could be challenging because sequences belong to an undefined vector space, hence measuring the similarity of two sequences itself is an expensive task. Trends in the past decade show that there is no longer significance improvement in the clock speed of processors. This indicates that parallel programming is necessary to be studied especially on metagenomics clustering where the size of data is growing steadily.
Chapter 3

Design and Algorithms

3.1 Challenges and Limitations

An enormous number of sequences is now available to be processed and this number is still growing steadily. The number of available cores is limited and so is memory space. The aim of the design in this project is to achieve the maximum speedup and scalability through the parallelisation of the sequential algorithm of single-linkage hierarchical clustering, given a limited amount of memory space.

By appreciating the knowledge from Amdahl’s law and Gustafson’s law, the following concepts stand as the foundation of the design of this project:

- The serial part of the program should be minimised as much as possible.
- The serial part of the program should be designed to grow as slowly as possible when the problem size increases.
- The parallel part of the program should be designed to run as independent as possible so that the overhead would not increase when a number of processors is added.

3.2 Concept of Hierarchical Clustering Algorithm

Hierarchical clustering can be obtained by first calculating the pairwise distance between each vertex followed by constructing a minimum spanning tree that later can be cut on a certain level to get a set of clusters. In single-linkage clustering, the exact distance between each vertex is not so important and would not affect the final result of the clustering since the tree will be cut if the distance between two vertices is greater than a desired resolution. To achieve more efficiency we could take advantage of this attribute by simply storing a boolean type to define the relation between vertices to represent whether they are connected.
Based on the definition above it follows that the number of calculations that needs to be performed on a group of sequences can be defined as the number of possible pairs of two sequences that can be picked from the input sequences. For \( n \) input sequences the number of pairwise distances is growing quadratically and can be defined as:

\[
C(n, 2) = \frac{n!}{(2!(n-2)!)} = \frac{1}{2}(n^2 - n)
\]  

(3.1)

For instance, clustering \( 10^3 \) sequences involves \( \approx 5 \times 10^5 \) pairwise distances while clustering \( 10^5 \) sequences would require the computation of \( \approx 5 \times 10^9 \) pairwise distances. To achieve a better performance some of the calculations can be skipped if they are confirmed as redundant based on the result of previous calculations.

In the course of sequence clustering, deciding the similarity between two sequences requires performing a pairwise global alignment which is quite expensive with complexity of \( \Theta(MN) \) when using the affine gap penalty, where \( M \) and \( N \) are the length of each sequences. In order to minimise calculation time, unnecessary alignments may be eliminated by a filtering scheme that involves a cheaper computation, such as:

**Comparing length of two sequences** If two sequences have a length difference more than \( d \) then it cannot have the edit distance of less than or equal to \( d \).

**Comparing the k-mer profiles of two sequences** Sequences that have more than \( 2dk \) different q-grams or k-mers of length \( k \) cannot have the edit distance of less than or equal to \( d \) [Mahé et al., 2014]. The length of \( k \) affects the cost of the k-mer profiles comparison because the number of possible k-mers \( (1-4^k) \) also increases when \( k \) is bigger. However the precision that can be obtained by using a different value of \( k \) remains open.

Figure 3.1 illustrates the 3-mer vector comparison of two sequences: ACTACGTAAG and ACTACGTAAG that we know have one single different nucleotide. This simulation follows the design in Swarm, where a q-gram profile vector contains either 0 or 1 to represent even or odd occurrences of a q-gram in the sequence. A more common practice is to assign 0 for no occurrence of the q-gram, and 1 for any occurrence(s) of the q-gram.

The first step of the comparison would be to generate the q-gram profile vectors for both sequences that register if the sequence has the odd number or the even number of 3-mer occurrences. Since there are four nucleotides A, C, T, G in a DNA sequence, then this vector has the length of \( 4^3 \) with value 0 for every even number of 3-mer occurrences, and value 1 for every odd number of 3-mer occurrences. No occurrence is registered as an even number of occurrences. Q-gram/k-mer distance of the two sequences can later be obtained by summing up the XOR result of each vector members. Two sequences with a k-mer distance more than \( 2dk \) cannot have less than or equal to \( d \) of edit distance. In
this example, the k-mer distance is not more than $2dk = 2(1)(3) = 6$ which means that the two sequences may have the edit distance of less than or equal to 1.

We have tested the following approaches during the development of this project, i.e.: (i) comparing the length of sequences and then proceed with comparing k-mer profiles, or (ii) only comparing k-mer profiles. As the result, approach (i) did not give a meaningful time improvement comparing to approach (ii), while in the algorithm of this project the comparison of k-mer profiles is always necessary for the economic search (see subsection 3.5.1.2). Therefore to rule out unnecessary alignments only the k-mer profiles comparison will be used in this project. K-mer profiles for all sequences are to be pre-calculated during the initialisation of the program and stored in the RAM so that the cost to compute the k-mer distance during the clustering is $O(1)$.

3.3 Parallelisation Strategies

There are a few approaches to the parallelisation that were explored during this work in order to find out a possible way to utilise multiple threads. The first approach that will be explained in section 3.4 is a distributed memory model where multiple parallel processes will each carry out an individual clustering on the partitions of sequences. Finally, a serial process will merge the results to conclude a result set for the entire dataset. The second approach in section 3.5 is a shared
memory model where the entire clustering process will be divided into a set of small tasks – referred as "row calculations" in the design – that can be carried out independently across multiple threads.

### 3.4 Distributed Memory Model: Parallel Clustering on Partitions of Dataset and Merging of Results

Sequences were partitioned into some groups that become the input sequences for each thread. Each thread would simultaneously cluster the input sequences using the same agglomerative algorithm from Swarm, producing their own result sets. Afterwards, these result sets can be merged either serially or in parallel to conclude a final answer for all of the sequences.

Let $t$ be the number of parallel threads and $N$ be the total input sequences, each partition of the distributed model would have $\frac{1}{t}N$ number of input sequences. Since the complexity of a serial hierarchical clustering is $\Theta(N^2)$, it means that the parallel clustering of each partition would take only $\frac{1}{t}T$, where $T$ is the time to cluster an $N$ number of sequences.

After each partition of the dataset has been individually clustered in each thread, the next step would be to merge the result sets from each thread into a single result set. Let clusters from first clusters set be $A = \{A_1, A_2, ..., A_m\}$ and clusters from second clusters set be $B = \{B_1, B_2, ..., B_n\}$.

For each member of the Cartesian product $A \times B = \{(a, b) | a \in A, b \in B\}$ where $|A \times B| = |A| \cdot |B|$ it shall be evaluated whether the pair should belong to the same cluster. If yes, then they will be merged. Once all combinations have been observed, all members of $A$ and $B$ will be put together as a single result set.

The following steps describe the process of the evaluation whether a pair of clusters should be merged: Let the first cluster be $A_1$ and let the second cluster be $B_1$. If there exists any pair of sequences where $d(a,b) \leq d, a \in A_1, b \in B_1$ then $A_1$ and $B_1$ belongs to the same cluster. In other words, two clusters should be merged if there exists any sequence in the first set that has the distance less than or equal to the clustering resolution with any sequence in the second set.

The following steps describe the shortcut for verifying that two clusters cannot be merged: Let $S_a$ be the seed of the first cluster, $S_b$ be the seed of the second cluster, and $d(S_a + S_b)$ be the edit distance between both sequences as illustrated on figure 3.2. Let $D_a$ be the maximum level of the subseeds from the first cluster and $D_b$ be the maximum level of the subseeds from the second cluster. If $d(S_a + S_b) > D_a + D_b$, then there cannot exist any sequences from each first and second clusters that are near with each other. On the contrary $d(S_a + S_b) \leq D_a + D_b$ does not mean the opposite, because the shape of a cluster is high dimensional and the variables in this equation do not belong to a metric space.
The result sets from each thread shall be merged with each other to a single clusters set. Up to this point the result is not yet final because whenever a cluster was merged with another cluster, it changes the structure of the cluster in a way that it might be connected with a cluster that was previously not merge-able. Therefore it is necessary to perform a recursive process as the last step that will compare these “merged clusters” with every other cluster in the clusters set until it is confirmed that there is no longer any merge-able clusters.

A set of code were written as a proof of concept to verify this idea, and it was run on a test dataset of around 1,500 sequences. The result shows that the part of the algorithm which merge the result sets from each parallel process was very inefficient because there were too many pairs of clusters that need to be evaluated especially in a natural biological dataset. Therefore, the work for this strategy was discontinued because of the high complexity of the merging problem.

### 3.5 Shared Memory Model: Parallel Calculations of Pairwise Distances

Given a set of input sequences \( S = \{1, 2, 3, \ldots, N\} \), hierarchical clustering can be obtained by calculating pairwise distances of each pair combinations of the sequences \( C = \{(1, 2), (1, 3), \ldots, (1, N), (2, 1), (2, 2), \ldots, (N, N)\} \). The complete comparison of all pairs that was mentioned before will be denoted in a form of "pairwise connection matrix" or "pairwise matrix" as shown on figure 3.3. We found this matrix to be useful as the thinking aid for breaking down the complete step of hierarchical clustering into small tasks so that they can be run in parallel.
Figure 3.3: Example of N sequences pairwise matrix

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>...</th>
<th>N</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>-</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>-</td>
<td>-</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>N</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Note: The dashed line between node 3 and 8 denotes a redundant calculation for \(c(3,8)\) because \(c(1,3) = 1\) and \(c(1,8) = 1\) from earlier comparisons. And so is \(c(10,N)\).

Figure 3.4: Sequence nodes correlated to pairwise matrix in previous figure.

Each cell in the matrix contains the information whether a sequence is connected to another sequence, \(c(a,b) = 0\) means that \(a\) and \(b\) are not connected while \(c(a,b) = 1\) means that \(a\) and \(b\) are connected. Two nodes are connected if they have the difference of less than or equal to a specified clustering resolution \(d\). The matrix can be used to build a set
of dendrograms (figure 3.4) that will lead to the final clustering result. A single row of the matrix represents the work of comparing a sequence to every other sequences with an ID greater than itself, which will be referred as "row calculation" in this paper, i.e., the task of calculating the distances between the row ID and all of the column IDs.

A pairwise matrix is always symmetrical since \( d(a, b) = d(b, a) \). Furthermore \( d(a, a) = 0 \), therefore the values on the main diagonal do not need to be computed. This means that a pairwise matrix consists of indeed \( \frac{1}{2}(n^2 - n) \) meaningful cells. Given a set of input sequences to be clustered \( S = 1, 2, 3, ..., 10 \), based on the pairwise matrix as presented in figure 3.4, a final clustering result set would be concluded as: \( R = \{1, 3, 8\}, \{2, 10\}, \{4, 6, 9\}, \{5\}, \{7\} \).

The calculation on each cell in the matrix as well as each row is independent and do not need to be chronological in order to get a correct result. With this kind of arrangement, each "row calculation" can be performed simultaneously across multiple threads. A complete hierarchical clustering process of \( N \) input sequences consists of \( (N - 1) \) row calculations. While each row calculation consists of \( (N - m) \) pairwise similarity comparisons, where \( m \) is the ID of the sequence in the input set ordered decreasingly by abundance. In other words, the row calculation for the first sequence would have \( (N - 1) \) pairwise similarity comparisons while the one for the last sequence would have zero comparison.

The parallelisation in this proposal is to create a thread pool consists of \( t \) number of threads. A thread organiser (or the main program) keeps track of the current row ID in the pairwise matrix computation. Whenever there is a thread that becomes available, it will be assigned to perform a row calculation for the current row ID. The thread organiser will then increment the current row ID and proceed to assign a row calculation for another idle thread. Once the row ID has reached the total number of input sequences, the process is considered as done, and all threads will be destroyed.

### 3.5.1 Omitting Redundant Calculation

The omitting redundant calculation is necessary especially for a large set of data and may lead to substantial performance improvement.

#### 3.5.1.1 Omitting Edges Outside of Minimum Spanning Tree

The computation of the pairwise distance between \( a \) and \( b \) will be redundant if it was already known from earlier calculations that \( a \) and \( b \) belong to the same cluster. For instance in the previous figure 3.4, node 1, 3 and 8 all belong to the same cluster. Since \( d(1, 3) = 1 \) and \( d(1, 8) = 1 \) are already sufficient to prove that \( \{1, 3, 8\} \) are a cluster, it would be redundant to calculate the pairwise distance between node 3 and node 8. Calculations that are similar to this example can be safely
omitted while they do not compromise the precision of the final result if they were computed anyway.

3.5.1.2 Economic Search on First Level Connections

Given the condition where node $a$ is connected with each node in $Z = \{b_1, b_2, \ldots, b_n\}$ or $c(a, b) = 1, b \in Z$ and the distance between $a$ and every node in $X = \{x_1, x_2, \ldots, x_n\}$ are more than twice the clustering resolution i.e. $d(a, x) > 2d, x \in X$, as shown on figure 3.5. Then the calculation of pairwise distances $d(b, x), b \in Z, x \in X$ can be safely omitted since they cannot be less than or equal to the resolution. This is because $d(a, x) \leq d(a, b) + d(b, x), b \in Z, x \in X$ and therefore $d(b, x) \geq d(a, x) - d(a, b), b \in Z, x \in X$.

In another word, it is sufficient to compare $Z = \{b_1, b_2, \ldots, b_n\}$ with $E = \{e_1, e_2, \ldots, e_n\}$ where $\forall e \in E$ $d(a, e) \leq 2d$.

The concept of economic search here is defined by following the strategy in Swarm where a subseed would only be compared to the amplicons which have no more than 2$d$ differences with the seed [Mahé et al., 2014]. In Swarm an identical economic search is performed without any limit to the level of connection, and therefore it may be more efficient than the economic search in this project. The limiting of the level of connection in this project is inevitable in order to keep the row calculations independent.

![Figure 3.5: Economic search on first level connections.](image)

3.5.1.3 Economic Search on Multiple Levels Connections

On the previous explanation, it was described how to skip the distance calculation on the set of first level connected sequences. The similar strategy may be further extended to skip the distance calculations on
the \( r \text{th} \) level of connected sequences if the distance of every sequence with the row sequence will be stored temporarily in the memory. On the first level connected sequences only the sequences with the distance no more than twice the resolution should be considered as candidates, and on the \( r \text{th} \) level of connected sequences only sequences within \((r+1)d\) needs to be considered, where \( d \) is the resolution.

In other words, let \( a \) be a sequence and \( Z_1 = \{b_1, b_2, ..., b_n\} \), \( d(a, b) \leq d \).

Let \( d(a, b) \leq d \), and furthermore \( d(a, b) \leq d \), then it is sufficient to compare \( Z_r = \{b_1, b_2, ..., b_n\} \) with \( E = \{e_1, e_2, ..., e_n\} \) where \( d(a, e) \leq (r+1)d \).

### 3.5.2 Forming Clusters Set

Building clusters based on the information from a pairwise matrix is rather simple:

- Iterate on every connection record in the pairwise matrix. Whenever a connection \((x, y)\) exists, perform a lookup on clusters set to see if \( x \) and \( y \) belong to any clusters.
  - If \( x \) and \( y \) belong to a same cluster, nothing needs to be done.
  - If \( x \) and \( y \) belong to two different clusters, then merge these clusters.
  - If \( x \) belongs to a cluster while \( y \) is new (does not belong to any cluster), then add \( y \) to the cluster. And vice versa.
  - If \( x \) and \( y \) are both new, create a new cluster and add both \( x \) and \( y \) to the new cluster.

- Upon completing the calculation of the pairwise matrix, there might exist some sequences that do not belong to any cluster. These sequences need to be identified and individually added as singleton clusters as the final step.

### 3.6 Data Structure Design

The data structure design is important since the problem size could be as big as 40 GB input of fasta file, or even more. The memory requirement for the program grows together with the problem size, meanwhile the amount of maximum memory is limited by the current available hardware on the market, operating system, or simply the budget of the user.

#### 3.6.1 Storing Cluster Sets

A cluster consists of a cluster ID and a set of cluster members. The clusters set will be stored as a mapping from the cluster ID to its cluster members. In this way, any cluster can be retrieved efficiently based on its cluster ID. Following the step to form a cluster as described above,
it is also necessary to store the reversed mapping information from a sequence member to its cluster ID. This is to facilitate the lookup of checking whether a sequence $x$ belongs to any cluster.

### 3.6.2 Storing of Pairwise matrix

The pairwise matrix data structure that was described earlier could take up a lot of space in memory. For an $N$ number of sequences and a boolean type or bit type for each of the matrix cell, it would take up to $N^2$ bits to store the pairwise matrix. It means that $10^5$ sequences require $10^{12}$ bit = 125 GB of memory, which is impractical.

Saving the connection matrix as a dynamic list of connected pair of sequences can save up a lot of space because the connection matrix is usually a sparse matrix with more cells containing zero values. However, the lookup on a list is usually as slow as $\mathcal{O}(N)$ or $\mathcal{O}(\log N)$ for a sorted list but this would not affect the execution time since our algorithm does not require any lookup on the pairwise matrix data.

To save more time and memory, there is no need to store the connection pairs if the step of building cluster sets can be done on the fly as a connection was detected. This is feasible since the way of forming clusters set was designed to be independent and the order of inserting pairs of connected sequences into the clusters set does not affect the clustering result. A synchronisation effort is necessary to prevent concurrent read/write phenomena on the global clusters set data. Only one thread at a time should be allowed to perform a lookup for any sequence on the clusters set, otherwise some out of date information could be retrieved and might lead to a wrong result set. For this reason, there will be a small overhead to synchronise the read/write access to the global cluster set data.

### 3.6.3 Storing Flags for Economic Search

As explained before, some calculations are redundant based on the triangle inequalities of the distance between three sequences and they may be eliminated without affecting the final result of the clustering. The economic search of first level connection can be achieved by storing a set of sequences that should be considered as the candidates for the row calculations of all connected sequences.

While for multiple level economic search, the distances from initial seed to all candidates must be stored in addition to the sequence itself so that these candidates can be grouped according to their level of connections with the row sequence. To minimise overhead, each thread will have their own temporary variables to store this information.
3.7 Implementation

3.7.1 Main Program

The main program starts by receiving a FASTA file, the value of \(d\), the depth of economic search, and the number of threads as the input. A preliminary check consists of the following steps:

- validation that there is no duplicate sequence header in the input file.
- validation that all sequences are comprised of valid nucleotides \(\{A, C, T, G\}\).
- validation that all sequences contain an abundance information and are sorted decreasingly by this abundance information.

Input sequences are stored in the global memory and the following attributes are generated for each sequence:

- sequence index, based on the order of the input, e.g. the first sequence has the index = 0, the second sequence has the index = 1, and so on.

Figure 3.6: Flowchart of the whole process.
- q-gram or k-mer profile (as described on section 2.4.4 and 3.2)
- length of the sequence
- visit flag, pre-defined as 0 (means not visited)

Followed by the initialisation of $t$ number of threads, each idle thread will be assigned with a row ID starting from 1 until (N-1) where N is the number of input sequences. Based on the submitted "depth" option, a brute-force approach, or a first-level economic search, or an n-level/multiple-level economic search may be performed inside the thread for the designated row ID. By default, the depth of the economic search is 1, which also means that no economic search will be performed, or a brute-force row calculation will be used. For $depth = 2$ it means that the economic search will be performed within $2d$ radius from the seed, or referred as first-level economic search. For $depth > 2$ it means that the economic search will be performed within $(depth)d$ radius from the seed, or referred as a multiple-level economic search.

The single unit of work that is performed by each thread is referred as the "row calculation". The operation for each method will be explained separately in the following subsections. The assigning of each row calculation to an idle thread shall go on until the row ID reaches (N-1) where the parallel work is considered as done and all threads may be destroyed. Up to this point, the construction of the clusters set is not yet final because some sequences that are not connected with any other sequences – referred as the singletons– were not included in the clusters set. A check will be performed for all sequences to verify whether the sequence belongs to any cluster, if not then a new cluster will be created for the sequence.

The flowchart of the entire process can be seen on figure 3.6.
Algorithm 1 Main Process

Input:
- \( T \): thread count
- \( file \): input FASTA file name and path
- \( d \): clustering resolution
- \( depth \): the level for economic search

Output:
\[ C = \{C_1, C_2, ..., C_m\} \]  
\( \triangleright \) clusters set

1: preconditions checking on submitted file
2: load all sequences \( S \) and generate k-mer profiles \( K \), save \( S \) and \( K \) into main memory
3: \( N \) ← number of sequences
4: initialise and start all threads \( T \)
5: \textbf{for} row ← 1 to \( N \) \textbf{do}
6: \hspace{1em} wait for available thread
7: \hspace{1em} if \( depth = 1 \) then
8: \hspace{2em} Call procedure brute_force_row_calculation (row, \( N \), \( d \), \( s \), \( k \))
9: \hspace{1em} else if \( depth = 2 \) then
10: \hspace{2em} Call procedure first_level_row_calculation (row, \( N \), \( d \), \( s \), \( k \))
11: \hspace{1em} else
12: \hspace{2em} Call procedure multiple_level_row_calculation (row, \( N \), \( d \), \( s \), \( k \), depth)
13: \hspace{1em} \textbf{end if}
14: \textbf{end for}
15: wait until all threads are finished
16: terminate and destroy all threads
17: \textbf{for} seq ← 1 to \( N \) \textbf{do}
18: \hspace{1em} if \( seq \notin C \) then \( \triangleright \) if sequence was not in any cluster
19: \hspace{2em} \( C_{m+1} \leftarrow seq \) \hspace{1em} \( \triangleright \) create new cluster and assign sequence
20: \hspace{2em} \( C \leftarrow C \cup C_{m+1} \) \hspace{1em} \( \triangleright \) add cluster to clusters set
21: \hspace{1em} \textbf{end if}
22: \textbf{end for}
23: \textbf{return} \( C \)

3.7.2 Row Calculation

Row calculations are individually performed inside each thread, one row calculation can be referred as one work unit where a thread may perform more than one work units in the entire clustering process. There are three options of row calculations that were defined for our program: brute-force search, first-level economic search, and multiple-level economic search.

3.7.2.1 Brute-Force Search

A brute-force search does not involve any skipping mechanism and it will simply go through all sequences from \( m + 1, m + 2, ..., N \) to identify a
set of connected sequences, where \( m \) is the row ID and \( N \) is the total input sequences. A connected sequence is a sequence that has the edit distance less than or equals to \( d \) with the row ID. To check whether a sequence is connected, the k-mer profile of both sequences will be compared. If there are more than \( 2dk \) differences then it means that the sequence is not connected, otherwise it will proceed with a pairwise alignment to verify if the edit distance is less than or equal to \( d \). Every time a connection was identified, algorithm 3 will be called to assign the pair of connected sequences into the correct cluster.

**Algorithm 2** Row Calculation for Brute-Force Search Method

**Procedure:** brute_force_row_calculation (row, \( N \), \( d \), \( S \), \( K \))

**Input:**
- \( row \): index of the row sequence
- \( N \): total number of input sequences
- \( d \): clustering resolution
- \( S \): all input sequences
- \( K \): K-mer profiles of all input sequences

1: for \( col \leftarrow row + 1 \) to \( N \) do
2: if difference\( (K_{row}, K_{col}) \leq d \) then
3: \( \triangleright K_n \) is the k-mer profile of sequence \( S_n \)
4: if alignment\( (S_{row}, S_{col}) \leq d \) then
5: FORM_CLUSTERS\( (row, col) \)
6: end if
7: end if
8: end for

### 3.7.2.2 Inserting Sequences into the Clusters Set

This procedure is called by submitting \( x \) and \( y \) as the arguments, where \( x \) and \( y \) are a pair of sequences which were identified to be connected or similar. Both sequences \( x \) and \( y \) or one of them will be inserted into the clusters set based on a set of rules described in section 3.5.2.

The global clusters set are denoted as \( C = C_1, C_2, ..., C_n \) in the pseudocode 3, where \( C_x = x_1, x_2, ..., x_m \) is a cluster consisting a set of \( m \) connected sequences. By the end of the clustering task, \( C \) would contain all of the input sequences grouped into \( n \) clusters. A singleton cluster is a cluster that contains only one member.

As mentioned in section 3.6.1, there are two maps that are used to store the clusters set information. The first map is implemented using an unordered map from Boost library as the container: \texttt{boost unordered_map<unsigned long int, cluster_info>}. It contains the mappings from the cluster ID to a cluster_info object. The cluster_info is a structure that consists of the cluster ID, and a list of sequences IDs of the cluster members (\texttt{std::vector<unsigned long int>}).
The second map is implemented as a primitive array `unsigned long int * member_stats`, where the array index represents the sequence ID and the value of the array represents the cluster ID. The reason to use a primitive array is to minimise the overhead based on the fact that the length of the array is fixed to the total input sequences.

To add a sequence to a cluster, the sequence ID is appended to the list of cluster members in the `cluster_info` followed by assigning `member_stats[sequence_id] = cluster_id`. The second map `member_stats` provides a fast access for checking whether a sequence is a member of a certain cluster.

The access to both of the maps must be protected by a lock since they are accessible to all of the threads and concurrent read or write operations on these objects may lead to the phenomenon of reading outdated information and undefined behaviour. The lock is obtained in the start of the update cluster procedure and released before exiting the procedure.

Algorithm 3 Assigning Sequences into the Clusters Set

Procedure: `form_clusters (x, y, C)`

Input:
- `x`: index of the first sequence
- `y`: index of the second sequence
- `C`: the global clusters set

1: wait and obtain lock for `C`
2: if `x ∈ C` and `y ∈ C` then ▶ Merge second cluster into first cluster
3: `C_x ← C_x ∪ C_y` where `C_x : x ∈ C_x, C_y : y ∈ C_y`
4: `C ← C − C_y`
5: else if `x ∉ C` and `y ∈ C` then ▶ Create new cluster for both sequences
6: `C_{m+1} ← \{x, y\}`
7: `C ← C ∪ C_{m+1}`
8: else if `x ∉ C` and `y ∉ C` then
9: `C_y ← C_y ∪ x` where `C_y : y ∈ C_y`
10: else if `x ∈ C` and `y ∉ C` then
11: `C_x ← C_x ∪ y` where `C_x : x ∈ C_x`
12: end if
13: release lock for `C`

3.7.2.3 First Level Connection Economic Search

A first level connection economic search includes the same step as a brute-force search with the extra work to save two groups of sequences in a temporary variable for the subsequent calculations, i.e. first level connected sequences, and candidates for the calculation of these sequences. In this option, one work unit of a thread consists of calculation of the row itself and row calculations of first level connected sequences. For instance if the row sequence `n` is connected with `Z =`
\{b_1, b_2, \ldots, b_m\}, and \(E = \{e_1, e_2, \ldots, e_p\} \) is a set of candidate sequences that each has distance not more than \(2d\) from \(n\), then a row calculation consists of the computation for rows: \(n\) and \(Z\), but sequences in \(Z\) will only be compared to candidate sequences \(E\) that have the ID greater than itself.

The flag for identifying whether row \(n\) has been visited is denoted as \(B_n\) in the pseudocode. The flag is necessary to prevent a row from being visited more than once because one row calculation may involve several non-sequential rows. Before performing a row calculation the procedure checks whether the flag for the row ID was already set as "visited". If yes, then it will exit the procedure because the row has been visited before, otherwise the flag will be updated to “1” and the procedure proceeds to the row calculation.
**Algorithm 4** Row Calculation for First Level Connection Economic Search Method

**Procedure:** first_level_row_calculation (row, N, d, s, k)

**Input:**
- **row**: index of the row sequence
- **N**: total number of input sequences
- **d**: clustering resolution
- **S**: all input Sequences
- **K**: K-mer profiles of all input sequences

1: if $B_{row} = true$ then
2:     return ► Exit this procedure because row has been visited before
3: else
4:     update $B_{row} ← true$
5: end if
6: for col ← row + 1 to N do
7:     $kmer\_diff ← difference(K_{row}, K_{col})$
8:     if $kmer\_diff ≤ d$ then
9:         $edit\_distance ← alignment(S_{row}, S_{col})$
10:        if $edit\_distance ≤ d$ then
11:            FORM_CLUSTER(row, col)
12:            $Z ← Z \cup col$  ► $Z$ is the set of all sequences connected to row
13:        else if $edit\_distance ≤ 2d$ then
14:            $E ← E \cup col$  ► $E$ is the set of all candidates for $Z$
15:        end if
16:     else if $kmer\_diff ≤ 2d$ then
17:         $E ← E \cup col$
18:     end if
19: end for
20: for all seq ∈ Z do
21:     if $B_{seq} = false$ then
22:         update $B_{seq} ← true$
23:         for all cand ∈ E such that cand > seq do
24:             ► Only need to consider candidates with ID greater than the sequence
25:             if $difference(K_{seq}, K_{cand}) ≤ d$ then
26:                 if $alignment(S_{seq}, S_{cand}) ≤ d$ then
27:                     FORM_CLUSTER(row, col)
28:                 end if
29:             end if
30:         end for
31:     end if
32: end for
3.7.2.4 Multiple Level Connection Economic Search

A multiple level connection economic search is the extension of previous procedure where an economic search of $(depth \geq 2)$ levels is to be performed, where $depth$ is the desired level of economic search. For each level, two groups of sequences: (i) a set of connected sequences, (ii) a set of candidate sequences, will be stored in the temporary memory.

Let connected sequences $Z = \{Z_1, Z_2, ..., Z_{depth}\}$ and candidates sequences $E = \{E_1, E_2, ..., E_{depth}\}$ be as defined in section 3.5.1.3. Then one row calculation consists of the computation for rows: $n$ and $Z_i$, but sequences in $Z_1$ will only be compared to candidate sequences $E_1$ that have the IDs greater than itself, $Z_2$ with $E_2$ and so on. Furthermore, the sequences in $E_1$ that were not connected with any of the sequences in $Z_1$ will be included in $E_2$ as the candidates for $Z_2$, and so on.

Algorithm 5 Row Calculation for Multiple Level Connection Economic Search Method

Procedure: multiple_level_row_calculation (row, N, d, s, k, depth)

Input:
row : index of the row sequence
N : total number of input sequences
d : clustering resolution
S : all input Sequences
K : K-mer profiles of all input sequences
depth : maximum level of economic search

1: if $B_{row} = true$ then
2: return
3: else
4: update $B_{row} ← true$
5: end if
6: for col ← row + 1 to N do
7: kmer_diff ← difference($K_{row}, K_{col}$)
8: if kmer_diff ≤ d then
9: edit_distance ← alignment($S_{row}, S_{col}$)
10: if edit_distance ≤ d then
11: FORM_CLUSTER($row, col$)
12: $Z_1 ← Z_1 \cup col$ ▶ $Z_1$ is the set of sequences connected to row
13: else if edit_distance ≤ (depth + 1)d then
14: $E_1 ← E_1 \cup col$ ▶ $E_1$ is the set of all candidates for $Z_1$
15: end if
16: else if kmer_diff ≤ (depth + 1)d then
17: $E_n ← E_n \cup col$ where $(n)d < d(row, col) \leq (n+1)d$
18: end if
19: end for

38
Row Calculation for Multiple Level Connection Economic Search Method (continued)

20: for $r \leftarrow 2$ to depth do
21:     for all $seq \in Z_r$ do
22:         if $B_{seq} = false$ then
23:             update $B_{seq} \leftarrow true$
24:                 for all $cand \in E_r$ such that $cand > seq$ do
25:                     if difference($K_{seq}, K_{cand}$) $\leq d$ then
26:                         if alignment($S_{seq}, S_{cand}$) $\leq d$ then
27:                             FORM_CLUSTER($row, col$)
28:                             $Z_{r+1} \leftarrow Z_{r+1} \cup col$
29:                             $\triangleright Z_{r+1}$ is the set of sequences connected to $Z_r$
30:                         else
31:                             $E_{r+1} \leftarrow E_{r+1} \cup col$
32:                             $\triangleright$ Candidates that are not close enough for this
33:                             level should be added to next level
34:                     end if
35:                 else
36:                     $E_{r+1} \leftarrow E_{r+1} \cup col$
37:                 end if
38:         end for
39:     end for
40: end for
Chapter 4

Methods and Materials

4.1 Programming Language and Libraries

P-swarm was developed in C++ version 98 and compiled with GCC – the GNU compiler collection. C++ is an extension to C and it has the same syntax as C. It is a multi-paradigm language which supports object oriented programming while does not enforce it. C++ comes with a set of standard library which consists of many useful implementations such as the container classes (vector, map, list, queue, etc).

Multithreading in p-swarm was implemented using POSIX thread (pthread) and mutex for shared memory mutual exclusion. Some codes for k-mer comparisons and global pairwise alignment that are very efficient and fast were reused from Swarm, with some efforts to migrate them to C++.

As the unit testint, the result set of p-swarm was compared with the result from Swarm to ensure that the program gives the correct results for different resolutions and numbers of threads.

4.2 Development Tools

4.2.1 Eclipse IDE for C/C++ Developers

Eclipse IDE has rich features for software development, such as: syntax checking, refactoring tools, code formatting, project and build, source code navigation, static code analysis and so on.

4.2.2 GDB for Program Debugging

GDB (GNU project debugger) is a debugging tool that can show us what the program was doing at the moment it crashed. GDB can be used to debug several types of languages such as Ada, Pascal, C, C++, Objective-C, and so on.
4.2.3 Valgrind for Memory Analysis

Valgrind is a dynamic analysis tool that provides a number of debugging and profiling tools. During the development of this project, the following tools from Valgrind were utilised to detect memory-related errors and bugs.

4.2.3.1 memcheck

Memcheck is the memory leak detector. This tool can generate a detailed report whenever memory leaks happen and it can also detect uses of uninitialised values. The stack trace generated by memcheck can tell us where the leaked memory was allocated and which kind of leaks happened, e.g. "definitely lost", "indirectly lost", or "possibly lost".

4.2.3.2 massif

Massif was used in this project to measure the amount of heap memory was consumed during runtime. Since C/C++ does not come with an automatic garbage collector, this tool could be very helpful in reducing the memory consumption and to pinpoint location of code with bad design that did not properly clean up object in the memory.

This tool records the snapshots of heap memory usage at every unit of time (measured by numbers of instructions executed) and it can also specify the type and originator of the objects that were occupying the heap.

Massif is usually being used together with ms_print that will aggregate the snapshots that were written by massif and summarise them into some tables and graphs that represent the growth of the heap across the entire timeline. A normal graph should show ideal growth of heap where the dead objects did not pile up.

4.2.4 Instruments from XCode for Profiling

Instruments is a performance analysis tool that was bundled as a part of Apple’s XCode development tools. It can be used to dynamically trace and profile OS X and iOS code, but can also be used to analyse C/C++ code in general. This project was using the time profiler template from Instruments in order to gain a deeper understanding on the workload and was useful to pinpoint the part of the code that was performing slow.

4.3 Hardware for Testing

Testings were performed on the Abel computer cluster – a shared resource for high-performance computing at the University of Oslo, hosted by USIT RIS group. Abel is comprised of more than 650 compute nodes each with minimum 64 GB RAM, 16 physical CPU cores – dual
Intel E5-2670 (Sandy Bridge) based running at 2.6 GHz. There are a few compute nodes that are equipped with up to 1 TB RAM, 32 physical CPU core – Intel E-4620 based running at 2.20 GHz\(^1\).

All compute nodes are running Linux, 64 bit of CentOS 6. The queue system in Abel is managed by SLURM workload manager.

### 4.4 Dataset for Testing

To observe the program’s performance, the tests were conducted on several datasets with an assorted number of amplicons and average nucleotides length as shown on table 4.1.

<table>
<thead>
<tr>
<th>Name</th>
<th>raw reads</th>
<th>unique raw reads</th>
<th>unique amplicons</th>
<th>average nucleotides</th>
</tr>
</thead>
<tbody>
<tr>
<td>EMP(^i)</td>
<td>(1.44 \times 10^{11})</td>
<td>(1.28 \times 10^{9})</td>
<td>(1.55 \times 10^{8})</td>
<td>117</td>
</tr>
<tr>
<td>BioMarKs(^ii)</td>
<td>(5.63 \times 10^{8})</td>
<td>(1.19 \times 10^{8})</td>
<td>(3.12 \times 10^{5})</td>
<td>381</td>
</tr>
<tr>
<td>Tara(^iii)</td>
<td>(5.74 \times 10^{10})</td>
<td>(1.20 \times 10^{9})</td>
<td>(9.50 \times 10^{6})</td>
<td>129</td>
</tr>
<tr>
<td>Greengenes(^iv)</td>
<td>(1.67 \times 10^{9})</td>
<td>(1.33 \times 10^{9})</td>
<td>(9.48 \times 10^{5})</td>
<td>1,404</td>
</tr>
</tbody>
</table>

\(^i\) Sample obtained from 60 studies of EMP project [Gilbert, Jansson, & Knight, 2014].
\(^ii\) Sample obtained from BioMarKs project [Logares et al., 2014].
\(^iii\) Sample obtained from TARA OCEANS project [Karsenti et al., 2011].
\(^iv\) Sample obtained from Greengenes database[DeSantis et al., 2006] version 13.5.

Table 4.1: Summary of Test Datasets

Sample processing, sequencing and core amplicon data analysis for EMP.fas were performed by the Earth Microbiome Project (www.earthmicrobiome.org) and all amplicon and metadata has been made public through the data portal (www.microbio.me/emp).

#### 4.4.1 Subsampling

The massive dataset such as EMP.fas will be sampled randomly to select a subset of the dataset as the representative. We are using vsearch [Flouri et al., 2015]\(^2\) for this purpose, for example with the following command:

```bash
vsearch --subsample EMP.fas --fraction 0.1 --output EMP_0.1.fas --seed 0
```

Running above’s command would randomly select 10% of the original sequence read **before dereplication**, and thus the result of the subsampling would not be comprised of exactly 10% of the entire unique sequences that could represent a more natural partition of a dataset’s subset.

\(^1\)http://www.uio.no/english/services/it/research/hpc/abel/more/index.html
\(^2\)subsampling is an unreleased feature of vsearch
4.5 Benchmarking with Other Tools – Command Lines and Software Versions

For benchmarking we have selected the following tools for comparison, listed with their version, source and commands.

- **swarm**
  version: 2.1.1
  source: https://github.com/torognes/swarm
  commands:
  ```
  ./swarm -t 16 BioMarKs.fas -o output -d 1 -a
  ./swarm -t 16 BioMarKs.fas -o output -d 3
  ```

- **CD-HIT**
  version: 4.6.1
  source: https://code.google.com/p/cdhit/
  commands:
  ```
  ./cd-hit -M 0 -T 0 -i BioMarKs.fas -o output -c 0.97
  ./cd-hit -M 0 -T 0 -i BioMarKs.fas -o output -c 0.95
  ```

- **usearch**
  version: 8.0
  source: http://www.drive5.com/usearch/download.html
  commands:
  ```
  ./usearch -cluster_fast BioMarKs.fas -threads 16 -id 0.97
  -centroids cent -uc uc
  ./usearch -cluster_fast BioMarKs.fas -threads 16 -id 0.95
  -centroids cent -uc uc
  ```

- **vsearch**
  version: 1.1.3
  source: https://github.com/torognes/vsearch
  commands:
  ```
  ./vsearch --cluster_fast BioMarKs.fas --threads 16 --id 0.97
  --centroids cent --uc uc
  ./vsearch --cluster_fast BioMarKs.fas --threads 16 --id 0.95
  --centroids cent --uc uc
  ```

- **DNACLUST**
  version: svn revision 54
  source: http://dnaclust.sourceforge.net/
  commands:
  ```
  ./dnaclust -t 16 --approximate-filter -s 0.97 BioMarKs.fas
  ./dnaclust -t 16 --approximate-filter -s 0.95 BioMarKs.fas
  ```

- **sumaclust**
  version: 1.0.01
  source: http://metabarcoding.org/sumatra
  commands:
Swarm is a single-linkage hierarchical clustering program while all the other tools are using heuristic and greedy clustering methods. These heuristic and greedy clustering methods use the terms "similarity" to group sequences into cluster, where “each sequence in the cluster must have a similarity above a given identity threshold (radius of a cluster) from the centroid”\(^3\). For these tools the definition of "cluster" is quite different with single-linkage hierarchical clustering, where the radius of a cluster is not defined and rather depends on the characteristics of the dataset. Therefore, the comparison will be done using two types of settings for each type of clustering tools: threshold 0.97 and 0.95 for the heuristic clustering tools; \(d = 1\) and \(d = 3\) for the hierarchical clustering tools. However due to the difference in cluster definition, these settings are not directly comparable.

\[^3\]http://drive5.com/usearch/manual/uclust_algo.html
Chapter 5

Results and Discussion

The program in this project is denoted as p-swarm and all tests in this chapter were executed using 5-mers for k-mer vector comparison. This chapter contains the presentation, evaluation, and discussion of the test result of p-swarm executed on different datasets and parameters, and also comparison of p-swarm with swarm and some of the popular heuristic clustering tools, i.e. CD-HIT, USEARCH, VSEARCH, DNAACLUST, and sumaclust.

5.1 Comparison of Speedup and Memory Consumption on Brute-Force, First-Level, and Multiple-Level

In the previous chapter about design, we proposed two economic search methods for omitting redundant calculations. Here we will present the results of running three different variants of p-swarm, namely: (i) without omitting calculations – labelled as brute-force, (ii) economic search on first level of connection – labelled as first-level, and (iii) economic search on multiple level of connection – labelled as multiple-level.

To observe the improvement of performance when more threads or processors are added, each variants of p-swarm were executed on BioMarKs dataset using $d = 1$ and $d = 3$ on several numbers of threads $t = 1, 2, 4, 8, 9, 12, 14, 16$. Table 5.1 shows the execution time of each variant while table 5.2 shows the comparison of speedup that was obtained on the each number of threads. Speedup is calculated by $speedup = T_1 / T_n$ where $T_1$ is the execution time if using a single thread, and $T_n$ is the execution time if using $n$ threads. In this experiment, multiple-search were set to perform the economic search up to the sixth level of connections.

As shown on the execution time comparison chart (see figure 5.1) for $d = 1$ the first-level method consistently performed better than the brute-force method with the improvement of 30% less execution time. When using the economic search on multiple level connections, p-swarm
performs at least twice better than the first-level method, and at least three times better than the brute-force method. The same result also applies for $d = 3$.

<table>
<thead>
<tr>
<th>threads</th>
<th>$d = 1$ time (sec)</th>
<th>$d = 3$ time (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>BF</td>
<td>FL</td>
</tr>
<tr>
<td>1</td>
<td>1,051.58</td>
<td>647.19</td>
</tr>
<tr>
<td>2</td>
<td>554.04</td>
<td>338.34</td>
</tr>
<tr>
<td>4</td>
<td>284.18</td>
<td>173.05</td>
</tr>
<tr>
<td>8</td>
<td>158.16</td>
<td>94.90</td>
</tr>
<tr>
<td>9</td>
<td>144.18</td>
<td>86.07</td>
</tr>
<tr>
<td>12</td>
<td>116.94</td>
<td>69.88</td>
</tr>
<tr>
<td>14</td>
<td>107.16</td>
<td>63.82</td>
</tr>
<tr>
<td>16</td>
<td>98.22</td>
<td>59.09</td>
</tr>
</tbody>
</table>

Table 5.1: Execution time of brute-force, first-level, multiple-level with various thread numbers

Figure 5.1: Evaluation of execution time on brute-force, first-level, multiple-level with various thread numbers for $d = 1$
Figure 5.2: Evaluation of execution time on brute-force, first-level, multiple-level with various thread numbers for $d = 3$

All methods gave consistent speedup when $d = 1$ where the maximum speedup that can be achieved with 16 threads is 11 times in average. By Amdahl’s law, this means that in theory the serial fraction of the program stands merely 3% (as shown on equation 5.1). For the value $d = 3$ the speedup of the multiple-level method is only 10 times while the speedup for the first-level method is almost 12 times. This indicates with a larger value of $d$ one could gain a better speed up if using the first-level method where the parallelism has a finer granularity.

$$T_n = T_1 \left( S + \frac{1}{n} (1 - S) \right)$$

$$\frac{1}{11} = S + \frac{1}{16} (1 - S)$$

$$S = 0.03$$

<table>
<thead>
<tr>
<th>threads</th>
<th>BF</th>
<th>FL</th>
<th>ML</th>
<th>BF</th>
<th>FL</th>
<th>ML</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
</tr>
<tr>
<td>2</td>
<td>1.90</td>
<td>1.91</td>
<td>1.92</td>
<td>1.98</td>
<td>1.95</td>
<td>1.89</td>
</tr>
<tr>
<td>4</td>
<td>3.70</td>
<td>3.74</td>
<td>3.77</td>
<td>4.37</td>
<td>3.84</td>
<td>3.59</td>
</tr>
<tr>
<td>8</td>
<td>6.65</td>
<td>6.82</td>
<td>6.98</td>
<td>7.11</td>
<td>7.35</td>
<td>6.71</td>
</tr>
<tr>
<td>9</td>
<td>7.29</td>
<td>7.52</td>
<td>7.71</td>
<td>7.80</td>
<td>8.09</td>
<td>7.36</td>
</tr>
<tr>
<td>12</td>
<td>8.99</td>
<td>9.26</td>
<td>9.64</td>
<td>9.69</td>
<td>10.20</td>
<td>8.95</td>
</tr>
<tr>
<td>14</td>
<td>9.81</td>
<td>10.14</td>
<td>10.72</td>
<td>10.44</td>
<td>11.03</td>
<td>9.55</td>
</tr>
<tr>
<td>16</td>
<td>10.71</td>
<td>10.95</td>
<td>11.39</td>
<td>10.73</td>
<td>11.85</td>
<td>9.89</td>
</tr>
</tbody>
</table>

Table 5.2: Speedups of brute-force, first-level, multiple-level with various thread numbers
Figure 5.3: Evaluation of speedup of brute-force, first-level, multiple-level with various thread numbers for $d = 1$

Figure 5.4: Evaluation of speedup of brute-force, first-level, multiple-level with various thread numbers for $d = 3$

Memory consumption is fairly stable along all methods where adding the number of threads would linearly increase the memory consumption. This is because using more threads requires the allocation of more temporary space to store batches of sequence alignment information. Theoretically both economic search methods — first-level and multiple-level — should require more temporary space to store next level connections and a set of candidate sequences (as explained in chapter 3), but test result shows only slight difference between the three methods. This is because the number of pairwise alignments that were performed in the brute-force method is more than the ones in the first-level and multiple-level, so that more memory was used for this occasion.
Table 5.3: Memory consumptions of brute-force, first-level, multiple-level with various thread numbers

<table>
<thead>
<tr>
<th>threads</th>
<th>$d = 1$ memory (MB)</th>
<th>$d = 3$ memory (MB)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>BF</td>
<td>FL</td>
</tr>
<tr>
<td>1</td>
<td>197</td>
<td>196</td>
</tr>
<tr>
<td>2</td>
<td>203</td>
<td>201</td>
</tr>
<tr>
<td>4</td>
<td>216</td>
<td>216</td>
</tr>
<tr>
<td>8</td>
<td>233</td>
<td>237</td>
</tr>
<tr>
<td>9</td>
<td>248</td>
<td>221</td>
</tr>
<tr>
<td>12</td>
<td>259</td>
<td>237</td>
</tr>
<tr>
<td>14</td>
<td>258</td>
<td>236</td>
</tr>
<tr>
<td>16</td>
<td>274</td>
<td>249</td>
</tr>
</tbody>
</table>

Figure 5.5: Evaluation of memory consumptions of brute-force, first-level, multiple-level with various thread numbers for $d = 1$

Figure 5.6: Evaluation of memory consumptions of brute-force, first-level, multiple-level with various thread numbers for $d = 3$
5.2 Comparison of Execution Times and Memory Consumptions on Datasets of Different Average Length

Both variants – first-level and multiple-level – were executed to cluster four datasets each with a different average length that each contains averagely 300,000 unique sequences. More details about the attribute of each dataset are presented on table 5.4. The BioMarKs dataset originally consists of the desired amount of unique sequences, while other datasets were subsampled to obtain a similar amount of unique sequences.

<table>
<thead>
<tr>
<th>dataset</th>
<th>total nt pre-drp.</th>
<th>total nt post-drp.</th>
<th>unique sequences</th>
<th>average length</th>
</tr>
</thead>
<tbody>
<tr>
<td>EMP_0.0006</td>
<td>$8.65 \times 10^7$</td>
<td>$4.06 \times 10^7$</td>
<td>352,211</td>
<td>115</td>
</tr>
<tr>
<td>TARA_0.01</td>
<td>$5.74 \times 10^9$</td>
<td>$4.54 \times 10^7$</td>
<td>354,020</td>
<td>128</td>
</tr>
<tr>
<td>BioMarKs</td>
<td>$5.63 \times 10^8$</td>
<td>$1.19 \times 10^8$</td>
<td>312,503</td>
<td>381</td>
</tr>
<tr>
<td>greengenes_0.35</td>
<td>$4.66 \times 10^8$</td>
<td>$4.66 \times 10^8$</td>
<td>331,831</td>
<td>1,404</td>
</tr>
</tbody>
</table>

\(a\) number in the file name represents the proportion that was subsampled from the dataset.
\(b\) total nucleotides before dereplication
\(c\) total nucleotides after dereplication

Table 5.4: Dataset of different average length

Previous experiments on BioMarKs have shown time improvement of multiple-level over first-level for \(d = 1\) and \(d = 3\), which apparently is not the case for datasets with short sequences such as EMP and TARA dataset. Table 5.5 shows that multiple-level starts to perform worse than first-level when \(d \geq 2\) for EMP dataset, and at \(d \geq 3\) for TARA dataset. These results correlate with the fact that EMP dataset is averagely shorter than TARA dataset.

Figure 5.7 shows the illustration of four level economic search where the row sequence is connected to two other sequences. The economic search area is the area that contains all sequences with the edit distance less than \(5d\) with the row sequence. Within this search area, sequences might or might not belong to the cluster that is connected with the row sequence. The cluster in this illustration is denoted with the red outline.

When the value of \(d\) is higher, the search area is also bigger and therefore it includes more sequences as the candidates. This yields to a higher overhead during the management of economic search flags. When the dataset is comprised of long sequences, it is still more cost-effective to use the multiple-level method since the cost to compare the k-mer profiles and to align longer sequences is even greater than the overhead that is caused by the economic search flags management. This is based on the fact that the time for performing pairwise alignment — denoted as \(O(N^2)\) — increases quadratically to the length of the
sequence. However for datasets with short sequences like TARA and EMP, the overhead turns out to be more expensive than the cost of k-mer comparison and sequence alignment. This premise is aligned with the findings in section 5.5, table 5.14 where the overhead is higher for a dataset with shorter sequences.

In other words, the advantage of using the multiple-level method becomes counterproductive when the cost to estimate and to calculate the proximity between two sequences is even cheaper than the cost to manage the flags for the economic search. The cost mentioned here is influenced by the size of the search area, where a larger $d$ would lead to a larger search area.

Note: This figure represent an example of one "row calculation" in the multiple-level economic search method.

Figure 5.7: Four Level Economic Search
We observed that the execution time for EMP dataset is longer than the one for TARA dataset although EMP dataset is comprised of shorter sequences than TARA dataset. This is related to the dataset characteristics where a more scattered dataset tends to require more time for clustering. A scattered dataset would contain more sequences that are singletons and do not form any cluster with the other sequences. The economic search is not applicable for these sequences since they do not have any connected members, therefore a full scan must be performed for each of the singletons. This explains why it requires a longer time to cluster these type of datasets. Table 5.6 shows the ratio of total sequences against the total clusters in the dataset, where a lower ratio means that the data is more scattered. Greengenes dataset having the lowest ratio appears to be the most scattered, followed by EMP, BioMarks, and TARA.

We can see from table 5.7 that the memory consumption correlates well with the number of total nucleotides after de-replication. Based on our calculation, the memory consumption is growing linearly to the input size — roughly 3.5 bytes was consumed for every nucleotide of the input dataset.
Table 5.7: Memory consumptions of dataset of different average length

<table>
<thead>
<tr>
<th>average length</th>
<th>d = 1 memory (MB)</th>
<th>d = 2 memory (MB)</th>
<th>d = 3 memory (MB)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>FL</td>
<td>ML</td>
<td>FL</td>
</tr>
<tr>
<td>115</td>
<td>149</td>
<td>162</td>
<td>137</td>
</tr>
<tr>
<td>128</td>
<td>188</td>
<td>167</td>
<td>160</td>
</tr>
<tr>
<td>381</td>
<td>249</td>
<td>273</td>
<td>245</td>
</tr>
<tr>
<td>1,404</td>
<td>1,203</td>
<td>1,210</td>
<td>1,204</td>
</tr>
</tbody>
</table>

5.3 Comparison of Execution Time and Memory on Datasets of Different Size

We have prepared ten different datasets that were subsampled from each TARA and EMP dataset, ranging from $1.09 \times 10^4$ to $9.50 \times 10^6$ unique sequences. All datasets were clustered with $d = 1$ and $d = 3$, each using the first-level method and the multiple-level method. More details of each dataset are listed on table 5.8.

Table 5.8: Datasets of different size

<table>
<thead>
<tr>
<th>dataset</th>
<th>total nt pre-drp.</th>
<th>total nt post-drp.</th>
<th>unique sequences</th>
<th>average length</th>
</tr>
</thead>
<tbody>
<tr>
<td>TARA_0.0001</td>
<td>5.74 x 10^6</td>
<td>1.39 x 10^6</td>
<td>1.09 x 10^4</td>
<td>128</td>
</tr>
<tr>
<td>TARA_0.001</td>
<td>5.74 x 10^7</td>
<td>8.07 x 10^6</td>
<td>6.29 x 10^4</td>
<td>128</td>
</tr>
<tr>
<td>TARA_0.01</td>
<td>5.74 x 10^8</td>
<td>4.55 x 10^7</td>
<td>3.54 x 10^5</td>
<td>128</td>
</tr>
<tr>
<td>TARA_0.1</td>
<td>5.74 x 10^9</td>
<td>2.48 x 10^8</td>
<td>1.93 x 10^6</td>
<td>128</td>
</tr>
<tr>
<td>TARA</td>
<td>5.74 x 10^10</td>
<td>1.20 x 10^9</td>
<td>9.50 x 10^6</td>
<td>129</td>
</tr>
<tr>
<td>EMP_0.0001</td>
<td>1.44 x 10^7</td>
<td>8.67 x 10^6</td>
<td>7.56 x 10^4</td>
<td>115</td>
</tr>
<tr>
<td>EMP_0.0005</td>
<td>7.22 x 10^7</td>
<td>3.47 x 10^7</td>
<td>3.01 x 10^5</td>
<td>115</td>
</tr>
<tr>
<td>EMP_0.001</td>
<td>1.44 x 10^8</td>
<td>6.25 x 10^7</td>
<td>5.42 x 10^5</td>
<td>115</td>
</tr>
<tr>
<td>EMP_0.005</td>
<td>7.22 x 10^8</td>
<td>2.41 x 10^8</td>
<td>2.08 x 10^6</td>
<td>116</td>
</tr>
<tr>
<td>EMP_0.01</td>
<td>1.44 x 10^9</td>
<td>4.29 x 10^8</td>
<td>3.69 x 10^6</td>
<td>116</td>
</tr>
</tbody>
</table>

| a | number in the file name represents the proportion that was subsampled from the dataset |
| b | total nucleotides before dereplication |
| c | total nucleotides after dereplication |

Table 5.8: Datasets of different size
<table>
<thead>
<tr>
<th></th>
<th>$d = 1$ time (sec)</th>
<th>$d = 3$ time (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>FL</td>
<td>ML</td>
</tr>
<tr>
<td>total seqs</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TARA_0.0001</td>
<td>0.16</td>
<td>0.18</td>
</tr>
<tr>
<td>TARA_0.001</td>
<td>1.22</td>
<td>1.17</td>
</tr>
<tr>
<td>TARA_0.01</td>
<td>44.38</td>
<td>30.73</td>
</tr>
<tr>
<td>TARA_0.1</td>
<td>1,456.20</td>
<td>661.86</td>
</tr>
<tr>
<td>TARA</td>
<td>35,025.41</td>
<td>13,885.99</td>
</tr>
<tr>
<td>EMP_0.0001</td>
<td>3.03</td>
<td>2.64</td>
</tr>
<tr>
<td>EMP_0.0005</td>
<td>49.12</td>
<td>44.52</td>
</tr>
<tr>
<td>EMP_0.001</td>
<td>162.28</td>
<td>142.26</td>
</tr>
<tr>
<td>EMP_0.005</td>
<td>2,238.11</td>
<td>1,175.14</td>
</tr>
<tr>
<td>EMP_0.01</td>
<td>6,997.51</td>
<td>5,334.96</td>
</tr>
</tbody>
</table>

Table 5.9: Execution times on datasets of different size

We can see that in this experiment the first-level method also outperforms the multiple-level method for $d = 3$ which is the same as what has happened in section 5.2 since TARA and EMP dataset are both comprised of short sequences. The run-time of the program increases quadratically along the growth of the problem size. This is expected as explained in section 3.2 that the number of operations that is needed for clustering is generally $\frac{1}{2}(n^2 - n)$. As shown on figure 5.8 and 5.9 the growth of the run-time for both methods and both value of $d$ is almost adjacent with the theoretical $O(n^2)$ line.

On the case of TARA dataset, the empirical run-time falls somewhat below the theoretical line while for EMP dataset the empirical runtime is very close to the theoretical $O(n^2)$ line. This result correlates well with the diversity of each dataset where a more diverse dataset would need more time to be clustered. As displayed on table 5.10, the subsamples of EMP dataset are more scattered than TARA dataset and its subsamples.
Figure 5.8: Empirical Run-time Growth for TARA dataset

Figure 5.9: Empirical Run-time Growth for EMP dataset
### Table 5.10: Clustering result of different problem size

<table>
<thead>
<tr>
<th>dataset</th>
<th>$d = 1$</th>
<th></th>
<th></th>
<th>$d = 3$</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>cluster</td>
<td>ratio *</td>
<td>cluster</td>
<td>ratio *</td>
<td></td>
</tr>
<tr>
<td>count</td>
<td>count</td>
<td></td>
<td>count</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TARA_0.0001</td>
<td>5,447</td>
<td>2.00</td>
<td>3,356</td>
<td>3.25</td>
<td></td>
</tr>
<tr>
<td>TARA_0.001</td>
<td>20,738</td>
<td>3.03</td>
<td>10,213</td>
<td>6.16</td>
<td></td>
</tr>
<tr>
<td>TARA_0.01</td>
<td>84,370</td>
<td>4.20</td>
<td>33,571</td>
<td>10.54</td>
<td></td>
</tr>
<tr>
<td>TARA_0.1</td>
<td>378,602</td>
<td>5.10</td>
<td>129,988</td>
<td>14.85</td>
<td></td>
</tr>
<tr>
<td>TARA</td>
<td>1,646,172</td>
<td>5.77</td>
<td>487,584</td>
<td>19.48</td>
<td></td>
</tr>
<tr>
<td>EMP_0.0001</td>
<td>52,148</td>
<td>1.45</td>
<td>33,064</td>
<td>2.29</td>
<td></td>
</tr>
<tr>
<td>EMP_0.0005</td>
<td>176,575</td>
<td>1.70</td>
<td>96,467</td>
<td>3.12</td>
<td></td>
</tr>
<tr>
<td>EMP_0.001</td>
<td>296,310</td>
<td>1.83</td>
<td>151,470</td>
<td>3.58</td>
<td></td>
</tr>
<tr>
<td>EMP_0.005</td>
<td>967,130</td>
<td>2.15</td>
<td>426,760</td>
<td>4.87</td>
<td></td>
</tr>
<tr>
<td>EMP_0.01</td>
<td>1,605,171</td>
<td>2.30</td>
<td>665,139</td>
<td>5.55</td>
<td></td>
</tr>
</tbody>
</table>

* ratio calculated as (unique sequences : cluster count).

Table 5.10: Clustering result of different problem size

### 5.4 Comparison of Execution Time and Memory Consumption on Different Values of $d$

Table 5.11 is a record of the time, memory and total clusters of BioMarKs dataset when clustered with $d = 1, 2, \ldots, 10$ using both first-level and multiple-level methods. We can see from figure 5.10 that the multiple-level method always performs better than the first-level method for all values of $d$. The growth of the execution time is almost linear for $1 \leq d \leq 9$. As shown on figure 5.11, the memory consumption for both methods is growing slowly as the value of $d$ increases. In average, the multiple-level method consumed slightly more memory than the first-level method.
<table>
<thead>
<tr>
<th></th>
<th>first-level</th>
<th>multiple-level</th>
</tr>
</thead>
<tbody>
<tr>
<td>d</td>
<td>time (sec)</td>
<td>memory (MB)</td>
</tr>
<tr>
<td>1</td>
<td>61.19</td>
<td>269</td>
</tr>
<tr>
<td>2</td>
<td>47.05</td>
<td>269</td>
</tr>
<tr>
<td>3</td>
<td>42.90</td>
<td>273</td>
</tr>
<tr>
<td>4</td>
<td>46.82</td>
<td>279</td>
</tr>
<tr>
<td>5</td>
<td>65.50</td>
<td>288</td>
</tr>
<tr>
<td>6</td>
<td>95.09</td>
<td>288</td>
</tr>
<tr>
<td>7</td>
<td>135.50</td>
<td>290</td>
</tr>
<tr>
<td>8</td>
<td>190.17</td>
<td>294</td>
</tr>
<tr>
<td>9</td>
<td>288.45</td>
<td>304</td>
</tr>
<tr>
<td>10</td>
<td>522.81</td>
<td>313</td>
</tr>
</tbody>
</table>

Table 5.11: Execution times and memory consumptions for different values of $d$ on BioMarKs dataset

Figure 5.10: Evaluation of execution time for different values of $d$ on BioMarKs dataset

Figure 5.11: Evaluation of memory consumption for different values of $d$ on BioMarKs dataset
The same set of experiment was performed on EMP_0.0005 dataset, as recorded on table 5.12. The difference of characteristics between BioMarKs and EMP_0.0005 datasets can be seen on figure 5.12 where the clustering on BioMarKs dataset produced less number of clusters comparing to ones on EMP_0.0005 dataset although both datasets comprised of the same number of unique sequences (BioMarKs consists of 312,503 sequences and EMP_0.0005 consists of 301,302 sequences).

The \textit{multiple-level} method not only uses more memory for \(2 \leq d \leq 8\), but it also took longer time than the \textit{first-level} method (see figure 5.13 and figure 5.14).

<table>
<thead>
<tr>
<th>(d)</th>
<th>\textit{first-level} time (sec)</th>
<th>\textit{first-level} memory (MB)</th>
<th>\textit{multiple-level} time (sec)</th>
<th>\textit{multiple-level} memory (MB)</th>
<th>\textit{total clusters}</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>48.97</td>
<td>124</td>
<td>43.47</td>
<td>124</td>
<td>176,575</td>
</tr>
<tr>
<td>2</td>
<td>38.44</td>
<td>121</td>
<td>43.37</td>
<td>173</td>
<td>126,605</td>
</tr>
<tr>
<td>3</td>
<td>33.85</td>
<td>166</td>
<td>64.06</td>
<td>234</td>
<td>96,467</td>
</tr>
<tr>
<td>4</td>
<td>32.81</td>
<td>120</td>
<td>97.55</td>
<td>270</td>
<td>74,543</td>
</tr>
<tr>
<td>5</td>
<td>45.38</td>
<td>174</td>
<td>124.36</td>
<td>275</td>
<td>58,342</td>
</tr>
<tr>
<td>6</td>
<td>86.67</td>
<td>222</td>
<td>160.71</td>
<td>288</td>
<td>46,208</td>
</tr>
<tr>
<td>7</td>
<td>139.29</td>
<td>250</td>
<td>195.52</td>
<td>288</td>
<td>36,962</td>
</tr>
<tr>
<td>8</td>
<td>214.61</td>
<td>297</td>
<td>233.34</td>
<td>294</td>
<td>29,936</td>
</tr>
<tr>
<td>9</td>
<td>337.2</td>
<td>318</td>
<td>316.07</td>
<td>324</td>
<td>24,477</td>
</tr>
<tr>
<td>10</td>
<td>621.56</td>
<td>311</td>
<td>533.4</td>
<td>314</td>
<td>22,120</td>
</tr>
</tbody>
</table>

Table 5.12: Execution times and memory consumptions of different values of \(d\) on EMP_0.0005 dataset
5.5 Analysis of CPU Usage and Time profile Samples

The CPU usage and time profile samples in this section were using Instruments from Apple Xcode version 6.2 on a Macbook Pro with Processor 2.6 GHz Intel Core i7, Memory 16 GB 1600 MHz DDR3.

The CPU usage graph for clustering BioMarKs dataset is presented in figure 5.15. During the initial step — between 00:00 to 00:01 — there was only a single thread that was active. After the initial step, all threads appeared to be busy with a bit of distortion which represent the idle time on some of the threads. Both graphs ended steeply which indicates that all threads finished their job at approximately the same time.
Figure 5.15: CPU usage graph of eight threads on clustering with $d = 1$ and $d = 3$

We have recorded the time profiling informations for $d = 1$ (see table 5.13) and $d = 3$ (see table 5.14) using the multiple-level method on BioMarKs dataset and EMP_0.0005 dataset. The time samples for light functions and system libraries were grouped together to summarise a simpler presentation. The records were listed in a descending order from the heaviest task to the lightest task.

<table>
<thead>
<tr>
<th>Proportion</th>
<th>Time (ms)</th>
<th>Task description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td><strong>BioMarks (443,084 ms)</strong></td>
</tr>
<tr>
<td>62.70%</td>
<td>277,814</td>
<td>comparing k-mer vector</td>
</tr>
<tr>
<td>24.80%</td>
<td>109,993</td>
<td>sequence alignment</td>
</tr>
<tr>
<td>8.30%</td>
<td>39,923</td>
<td>light functions and system libraries</td>
</tr>
<tr>
<td>2.50%</td>
<td>11,074</td>
<td>fetching sequence by ID</td>
</tr>
<tr>
<td>1.60%</td>
<td>7,147</td>
<td>backtrack in sequence alignment</td>
</tr>
<tr>
<td>0.10%</td>
<td>481</td>
<td>read sequences from file</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>EMP_0.0005 (731,727 ms)</strong></td>
</tr>
<tr>
<td>85.60%</td>
<td>626,359</td>
<td>comparing k-mer vector</td>
</tr>
<tr>
<td>10.38%</td>
<td>76,105</td>
<td>light functions and system libraries</td>
</tr>
<tr>
<td>3.30%</td>
<td>24,253</td>
<td>fetching sequence by ID</td>
</tr>
<tr>
<td>0.70%</td>
<td>5,609</td>
<td>sequence alignment</td>
</tr>
<tr>
<td>0.01%</td>
<td>530</td>
<td>backtrack in sequence alignment</td>
</tr>
<tr>
<td>0.01%</td>
<td>333</td>
<td>read sequences from file</td>
</tr>
</tbody>
</table>

Table 5.13: Time profile summary for $d = 1$
We can see from both tables that the k-mer comparisons and the pairwise alignments on BioMarKs dataset appear to be the heaviest tasks in the clustering process while other administrative tasks occupy only a small portion of the entire run-time. The administrative tasks comprised of the following functions:

- reading sequences from file
- generating the k-mer profile of all sequences
- lookup and assigning sequence into existing cluster, merging clusters, or creating new ones
- fetching and setting the row visit flag
- other functions from system libraries such as: creating and destroying objects, adding elements to `std::vector`, and so on.

For BioMarKs dataset, the proportion of total time consumption of k-mer comparisons and the pairwise alignments are not the same for \( d = 1 \) and \( d = 3 \). For \( d = 1 \) more time was spent on k-mer comparison while for \( d = 3 \) more time was spent on sequence alignment.

This is not the case for EMP dataset where most time was spent on k-mer comparisons and administrative tasks. Sequence alignment contributes to merely 3.30% and 2.80% of the total runtime. This indicates that using the multiple-level search for datasets with short sequences may lead into a higher overhead that is counterproductive to the performance.

To understand more about these numbers, we have also recorded the quantity of k-mer comparisons and pairwise alignments that were

<table>
<thead>
<tr>
<th>Proportion</th>
<th>Time (ms)</th>
<th>Task description</th>
</tr>
</thead>
<tbody>
<tr>
<td>43.50%</td>
<td>163,069</td>
<td>sequence alignment</td>
</tr>
<tr>
<td>43.20%</td>
<td>162,128</td>
<td>comparing k-mer vector</td>
</tr>
<tr>
<td>8.40%</td>
<td>35,448</td>
<td>light functions and system libraries</td>
</tr>
<tr>
<td>3.40%</td>
<td>13,272</td>
<td>backtrack in sequence alignment</td>
</tr>
<tr>
<td>1.40%</td>
<td>5,551</td>
<td>fetching sequence by ID</td>
</tr>
<tr>
<td>0.10%</td>
<td>422</td>
<td>read sequences from file</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Proportion</th>
<th>Time (ms)</th>
<th>Task description</th>
</tr>
</thead>
<tbody>
<tr>
<td>60.40%</td>
<td>542,122</td>
<td>comparing k-mer vector</td>
</tr>
<tr>
<td>34.69%</td>
<td>311,892</td>
<td>light functions and system libraries</td>
</tr>
<tr>
<td>2.80%</td>
<td>25,372</td>
<td>sequence alignment</td>
</tr>
<tr>
<td>1.30%</td>
<td>12,198</td>
<td>fetching sequence by ID</td>
</tr>
<tr>
<td>0.80%</td>
<td>7,193</td>
<td>backtrack in sequence alignment</td>
</tr>
<tr>
<td>0.01%</td>
<td>308</td>
<td>read sequences from file</td>
</tr>
</tbody>
</table>

Table 5.14: Time profile summary for \( d = 3 \)
carried out during the clustering operation on the \textit{first-level} method and the \textit{multiple-level} method (see table 5.15 and table 5.16).

<table>
<thead>
<tr>
<th>K-mer comparison</th>
<th>Pairwise alignment</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>d</th>
<th>Total amount</th>
<th>Percentage(^*)</th>
<th>Total amount</th>
<th>Per 10(^4) (^**)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

BioMarks

<table>
<thead>
<tr>
<th>d</th>
<th>Total amount</th>
<th>Percentage(^*)</th>
<th>Total amount</th>
<th>Per 10(^4) (^**)</th>
</tr>
</thead>
<tbody>
<tr>
<td>d = 1</td>
<td>2.72 \times 10^{10}</td>
<td>55.73</td>
<td>3.41 \times 10^6</td>
<td>1.25</td>
</tr>
<tr>
<td>d = 3</td>
<td>8.56 \times 10^9</td>
<td>17.54</td>
<td>9.69 \times 10^6</td>
<td>11.32</td>
</tr>
<tr>
<td>d = 5</td>
<td>5.39 \times 10^9</td>
<td>11.04</td>
<td>1.95 \times 10^7</td>
<td>36.20</td>
</tr>
</tbody>
</table>

EMP_0.0005

<table>
<thead>
<tr>
<th>d</th>
<th>Total amount</th>
<th>Percentage(^*)</th>
<th>Total amount</th>
<th>Per 10(^4) (^**)</th>
</tr>
</thead>
<tbody>
<tr>
<td>d = 1</td>
<td>3.24 \times 10^{10}</td>
<td>71.39</td>
<td>3.07 \times 10^5</td>
<td>0.09</td>
</tr>
<tr>
<td>d = 3</td>
<td>2.05 \times 10^9</td>
<td>45.22</td>
<td>4.90 \times 10^5</td>
<td>2.39</td>
</tr>
<tr>
<td>d = 5</td>
<td>1.46 \times 10^9</td>
<td>29.94</td>
<td>3.19 \times 10^7</td>
<td>21.83</td>
</tr>
</tbody>
</table>

\(^*\) Ratio against total edges of the dendrogram in percent.
\(^**\) Ratio against total amount of k-mer comparisons in per ten thousand.

Table 5.15: Operations count of \textit{first-level} method on BioMarKs and EMP_0.0005 dataset

<table>
<thead>
<tr>
<th>K-mer comparison</th>
<th>Pairwise alignment</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>d</th>
<th>Total amount</th>
<th>Percentage(^*)</th>
<th>Total amount</th>
<th>Per 10(^4) (^*)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

BioMarks

<table>
<thead>
<tr>
<th>d</th>
<th>Total amount</th>
<th>Percentage(^*)</th>
<th>Total amount</th>
<th>Per 10(^4) (^*)</th>
</tr>
</thead>
<tbody>
<tr>
<td>d = 1</td>
<td>12.70 \times 10^9</td>
<td>26.02</td>
<td>1.17 \times 10^6</td>
<td>0.92</td>
</tr>
<tr>
<td>d = 3</td>
<td>5.52 \times 10^9</td>
<td>11.30</td>
<td>1.93 \times 10^6</td>
<td>3.51</td>
</tr>
<tr>
<td>d = 5</td>
<td>4.14 \times 10^9</td>
<td>8.48</td>
<td>5.20 \times 10^6</td>
<td>12.55</td>
</tr>
</tbody>
</table>

EMP_0.0005

<table>
<thead>
<tr>
<th>d</th>
<th>Total amount</th>
<th>Percentage(^*)</th>
<th>Total amount</th>
<th>Per 10(^4) (^*)</th>
</tr>
</thead>
<tbody>
<tr>
<td>d = 1</td>
<td>2.86 \times 10^{10}</td>
<td>62.92</td>
<td>2.97 \times 10^5</td>
<td>0.10</td>
</tr>
<tr>
<td>d = 3</td>
<td>1.73 \times 10^{10}</td>
<td>38.05</td>
<td>4.04 \times 10^5</td>
<td>2.34</td>
</tr>
<tr>
<td>d = 5</td>
<td>1.14 \times 10^{10}</td>
<td>23.32</td>
<td>2.20 \times 10^7</td>
<td>19.31</td>
</tr>
</tbody>
</table>

\(^*\) Ratio against total edges of the dendrogram in percent.
\(^*\) Ratio against total amount of k-mer comparisons in per ten thousand.

Table 5.16: Operations count of \textit{multiple-level} method on BioMarKs and EMP_0.0005 dataset

BioMarKs dataset comprised of 312,503 unique sequences which means \(48.83 \times 10^9\) of pairwise distance computations are needed if clustering is to be performed with the \textit{brute-force} method. The \textit{first-level} economic search was able to omit more than 40\% of these computations while the \textit{multiple-level} method was able to omit more than 70\% of the distance computations.
A higher value of $d$ leads to less pairwise distance computations but it also yields more false positives in k-mer comparisons. More false-positives lead to more pairs of sequences that need to be verified by the expensive pairwise alignment. From table 5.15 and 5.16 it follows that k-mer comparison filtering method appears to be less effective for a higher value of $d$ and for the dataset with longer sequences.

Increasing $k$ will indeed lead to the exponential growth of complexity since the number of operations that needs to be performed is influenced by the length of the k-mer profile vector, which is $4^k$ for DNA sequence string. However, a higher $k$ value would produce more precision because longer k-mers are less likely to match by coincidence. For instance in a DNA sequence there are $4^3 = 64$ possible 3-mers, 256 possible 4-mers, 1024 possible 5-mers, and so on. The increase of $d$ might lead to less precision in estimating the edit distance because it increases the value of the allowed different bits, i.e. for 5-mer and $d = 1$, it allows $2dk = 10$ differences, and $d = 3$ allows 30 differences.

5.6 Comparison with Swarm and Other Heuristic Tools

5.6.1 Selection of Tools and Dataset

The program in this thesis is denoted as p-swarm and categorised as a hierarchical method tool, same as swarm. For the heuristic tools, we have selected CD-HIT, USEARCH, VSEARCH, DNACLUST, and sumaclust. Each of the programs was executed to cluster five different datasets ordered by the increasing number of unique sequences: BioMarKs, greengenes, TARA_0.1, EMP_0.01, and TARA. Two types of settings were used for these programs: $d = 1$ and $d = 3$ for the hierarchical tools; $id = 0.97$ and $id = 0.95$ for the heuristic tools. P-swarm was running the multiple-level method up to the sixth level to cluster all of the datasets, except in the case of $d = 3$ on EMP and TARA dataset it was running the first-level method. All of the tools support multi-threading and were using 16 threads in this experiment. More details about the version and the command line syntax when running each tool can be seen in section 4.5.

As mentioned in section 2.4.3, swarm is an exact, unsupervised, and single-linkage clustering method. Swarm and p-swarm produce exactly the same result set since they have the same concept and algorithm, and differ only on the parallelisation technique. Parallelisation in swarm involves simultaneous k-mer comparisons and pairwise alignments on multiple sequences while the workflow of the clustering is executed chronologically [Mahé et al., 2014].
5.6.2 Results of Benchmarking with Other Programs

The time, memory and clusters count were recorded for each execution as shown on table 5.17, 5.18 and 5.19. The 32-bit version of USEARCH which is free, can only handle up to 4GB of memory, therefore the record for TARA dataset is not available for USEARCH since the memory requirement to cluster this dataset is beyond the limit of the 32-bit version. While for swarm, to cluster TARA dataset with $d = 3$ may take a very long time (more than 3 days), hence the record is not presented here. The same applies for sumaclust for the execution record of greengenes dataset and TARA dataset.

The special algorithm from Swarm for clustering with $d = 1$ has the linear complexity where it took the least time to cluster all of the datasets, especially for the very large datasets the time difference with all other tools becomes very significant. While for $d = 3$ p-swarm performed averagely 10 times better than swarm. Swarm and p-swarm produce the same amount of clusters since they are both single-linkage hierarchical.

We can see from table 5.17 that DNACLUST and sumaclust on average spent the longest time to cluster each of the datasets. CDHIT and VSEARCH both have nearly the same performance, while USEARCH seems to perform better on greengenes dataset, which is comprised of long sequences. Memory consumption was fairly comparable among all tools, except for DNACLUST and sumaclust which consumed more than twice of the memory comparing to other heuristic tools on most of the datasets.

Memory consumption of p-swarm is either lower than – or comparable with all of the other tools. Apart from that p-swarm is also able to cluster all of the datasets for both settings $d = 1$ and $d = 3$ within a competitive runtime while preserving the quality of being the exact and unsupervised clustering. For example p-swarm is able to cluster the TARA dataset which comprises nearly 10 million unique sequences in less than four hours for $d = 1$ and less than three hours for $d = 3$, while for DNACLUST it took almost one day for $id = 0.97$ and eight hours for $id = 0.95$. As for swarm, it takes only five minutes to cluster the dataset with $d = 1$, but for $d = 3$ it took more than 3 days (the exact time was not recorded in this experiment).
<table>
<thead>
<tr>
<th>Program</th>
<th>BioMarKs</th>
<th>greengenes</th>
<th>TARA_0.1</th>
<th>EMP_0.01</th>
<th>TARA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hiarchical</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>p-swarm</td>
<td>30</td>
<td>795</td>
<td>662</td>
<td>5,335</td>
<td>13,887</td>
</tr>
<tr>
<td>swarm</td>
<td>14</td>
<td>215</td>
<td>63</td>
<td>123</td>
<td>308</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Heuristic</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CD-HIT</td>
<td>56</td>
<td>1,457</td>
<td>5,030</td>
<td>2,174</td>
<td>10,824</td>
</tr>
<tr>
<td>USEARCH</td>
<td>71</td>
<td>921</td>
<td>4,955</td>
<td>10,719</td>
<td>N/A**</td>
</tr>
<tr>
<td>VSEARCH</td>
<td>46</td>
<td>1,898</td>
<td>1,518</td>
<td>3,262</td>
<td>19,255</td>
</tr>
<tr>
<td>DNACLUST</td>
<td>154</td>
<td>2,784</td>
<td>3,670</td>
<td>34,586</td>
<td>69,599</td>
</tr>
<tr>
<td>sumaclust</td>
<td>51</td>
<td>477</td>
<td>N/A*</td>
<td>4,210</td>
<td>N/A*</td>
</tr>
</tbody>
</table>

* not available due to very long execution time.
** not available due to memory limitation of the free version of USEARCH.

Table 5.17: Execution Time (Seconds) Comparison with Other Clustering Tools
<table>
<thead>
<tr>
<th>Program</th>
<th>BioMarKs</th>
<th>greengenes</th>
<th>TARA_0.1</th>
<th>EMP_0.01</th>
<th>TARA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hierarchical</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>d = 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>d = 3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>p-swarm</td>
<td>273</td>
<td>268</td>
<td>2,294</td>
<td>2,278</td>
<td>752</td>
</tr>
<tr>
<td>swarm</td>
<td>193</td>
<td>231</td>
<td>2,084</td>
<td>2,214</td>
<td>486</td>
</tr>
<tr>
<td>Heuristic</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>id = 0.97</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>id = 0.95</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CD-HIT</td>
<td>761</td>
<td>759</td>
<td>2,353</td>
<td>2,325</td>
<td>1,745</td>
</tr>
<tr>
<td>USEARCH</td>
<td>224</td>
<td>190</td>
<td>2,389</td>
<td>1,893</td>
<td>775</td>
</tr>
<tr>
<td>VSEARCH</td>
<td>447</td>
<td>448</td>
<td>4,762</td>
<td>4,766</td>
<td>1,182</td>
</tr>
<tr>
<td>DNACLUST</td>
<td>1,757</td>
<td>1,753</td>
<td>13,834</td>
<td>13,834</td>
<td>2,910</td>
</tr>
<tr>
<td>sumaclust</td>
<td>997</td>
<td>998</td>
<td>N/A*</td>
<td>N/A*</td>
<td>5,193</td>
</tr>
</tbody>
</table>

* not available due to very long execution time.
** not available due to memory limitation of the free version of USEARCH.

Table 5.18: Memory (MB) Comparison with Other Clustering Tools
<table>
<thead>
<tr>
<th>Program</th>
<th>BioMarKs</th>
<th>greengenes</th>
<th>TARA_0.1</th>
<th>EMP_0.01</th>
<th>TARA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hierarchical</td>
<td>$d = 1$</td>
<td>$d = 3$</td>
<td>$d = 1$</td>
<td>$d = 3$</td>
<td>$d = 1$</td>
</tr>
<tr>
<td>p-swarm</td>
<td>89,745</td>
<td>34,456</td>
<td>769,116</td>
<td>539,282</td>
<td>378,602</td>
</tr>
<tr>
<td>swarm</td>
<td>89,745</td>
<td>34,456</td>
<td>769,116</td>
<td>539,282</td>
<td>378,602</td>
</tr>
<tr>
<td>Heuristic</td>
<td>$id = 0.97$</td>
<td>$id = 0.95$</td>
<td>$id = 0.97$</td>
<td>$id = 0.95$</td>
<td>$id = 0.97$</td>
</tr>
<tr>
<td>CD-HIT</td>
<td>17,745</td>
<td>12,145</td>
<td>85,345</td>
<td>50,906</td>
<td>246,744</td>
</tr>
<tr>
<td>USEARCH</td>
<td>39,617</td>
<td>14,897</td>
<td>127,037</td>
<td>71,152</td>
<td>382,536</td>
</tr>
<tr>
<td>VSEARCH</td>
<td>23,121</td>
<td>14,652</td>
<td>88,278</td>
<td>53,457</td>
<td>284,742</td>
</tr>
<tr>
<td>DNAACLUST</td>
<td>25,605</td>
<td>17,750</td>
<td>119,196</td>
<td>71,091</td>
<td>401,701</td>
</tr>
<tr>
<td>sumaclust</td>
<td>21,208</td>
<td>13,833</td>
<td>N/A*</td>
<td>N/A*</td>
<td>340,686</td>
</tr>
</tbody>
</table>

* not available due to very long execution time.
** not available due to memory limitation of the free version of USEARCH.

Table 5.19: Clusters Count Comparison with Other Clustering Tools
Chapter 6

Conclusion and Future Work

6.1 Conclusion

We have presented and implemented a parallelised single-linkage hierarchical clustering program named “p-swarm” with the options to run three different variants: brute-force, first-level and multiple-level. This program is available for free with GNU Affero GPL and can be downloaded from the public code repository – GitHub\(^1\). P-swarm was developed as the parallelised version of “swarm”, a robust and fast single-linkage hierarchical clustering for amplicon-based study. Inheriting the same clustering quality as swarm’s, our program runs averagely ten times faster than swarm when tested with 16 threads and \(d = 3\) on several selected datasets. This result has opened up the possibility to hierarchically cluster a larger volume of dataset, for instance, it only takes a few hours for p-swarm to cluster TARA dataset which comprises almost 10 million amplicons.

We have tested the program on several datasets with different average lengths, using different values of \(d\) and we have also observed the growth of time and memory as the problem size increases. The speedup of each variant of p-swarm was 11 times in average when running with 16 threads, which shows a good scalability.

When p-swarm was tested on different problem sizes, it was observed that the growth of the runtime for first-level method and multiple-level method each matches the theoretical complexity of the algorithm, which is \(O(n^2)\).

The results show that the memory consumption of our program has always been stable for all variants and all experiment sets where it is linear to the input size. In average p-swarm consumes a compatible amount of memory, if not less than swarm and the other heuristic tools. This result demonstrates an efficient utilisation of memory in our program.

\(^1\)https://github.com/mimitantono/p-swarm
Based on the result of the tests it was found out that the economic search could be less effective for clustering datasets that are more scattered. This is caused by the large number of singleton sequences in this type of datasets that each requires a full scan.

It is recommended to cluster datasets with long sequences using the \textit{multiple-level} method for any value of $d$, while datasets with short sequences may have better performance with the \textit{multiple-level} only for $d \leq 2$. This is based on the fact that the cost to calculate the distance between two short sequences is cheaper than the cost to manage the economic search on a large area (a higher $d$ leads to a larger search area). For this reason, it is recommended to use the \textit{first-level} method for clustering a short sequences dataset especially when $d > 2$.

Based on the profiling information that was collected on p-swarm, we observed that there is a need to achieve a balance between the amount of false positives that were produced by the k-mer comparison and the cost of performing the k-mer comparison itself in order to maximise the performance of the program. A higher value of $k$ may yield less false positives but more costly k-mer comparisons, meanwhile a higher number of false positives would eventually lead to more numbers of expensive pairwise alignments.

We have also compared the runtime and memory consumption of p-swarm with swarm and five other heuristic tools — CD-HIT, USEARCH (32-bit version), VSEARCH, DNACLUST, and sumaclust — on five different datasets with two types of settings. It was observed that p-swarm was able to handle all of the datasets for both settings, where sumaclust and swarm (for $d = 3$) were not able to cluster within a reasonable amount of time. TARA dataset is the biggest dataset in this experiment that consists of 10 million amplicons. The runtime of p-swarm was found out to be approximately 10 times faster than swarm for $d = 3$ and comparable with other heuristic clustering tools. This result suggests that p-swarm is able to cluster large datasets in a competitive runtime while preserving the quality of being single-linkage hierarchical.

\section*{6.2 Recommendation for Future Work}

Despite the successful outcome of the parallelisation approach that has been presented in this thesis, it may still be improved in several ways as following.

\subsection*{6.2.1 Extending the Code to Run on Multiple-Nodes Environment}

There are still potentials to further improve the speedup by increasing the number of processors. The parallel computing in this thesis is implemented using a low-level API — POSIX threads or pthreads that does not support multiple-nodes computer cluster. Consequently this
program can utilise only the CPU cores in a single node, where most processor families support up to a limited number of CPU cores, e.g. 18 cores for Xeon E5-4669 v3\(^2\) which was recently released in June 2015.

Based on the reason above, replacing the pthreads implementation with an API that supports multi-platform shared memory such as OpenMP\(^3\) or OpenMPI\(^4\) may significantly improve the scalability of p-swarm as it is easier and more economic to get multiple cores on a multiple-nodes computer cluster. However using these sophisticated frameworks also implicates more challenges in managing the shared and local memory not to mention more efforts in debugging during the implementation. Apart from that one would need to consider that the performance of the program may be limited by the communication speed between the nodes, therefore the design of the workflow should be made as asynchronous as possible.

6.2.2 Extending the Algorithm to a Distributed Memory Model

Alternatively the algorithm that was described and implemented in this project can also be extended to a distributed shared memory model where there is no communication needed between one node and another. The brute-force method for the algorithm does not require a shared memory since each row calculation is, in fact, independent and does not need to be executed sequentially. While the economic search can be implemented in the distributed memory model by dividing the input sequences into a set of partitioned datasets that are each assigned to the distributed parallel processor. Each parallel processor is only allowed to perform economic searches on the sequences within the assigned partition. This is to ensure that every row calculation will be processed only once.

The result of the row calculations on each parallel processors is a list containing pairs of connected sequences as described in section 3.6.2. These lists are to be processed in the final step that could be either serial or parallel to build a final result set, followed by finding the singleton sequences.

A distributed memory model like this is suitable for the implementation with a big data distributed processing frameworks like Apache Hadoop MapReduce\(^5\) or other similar technology. For instance, each row calculation can be implemented as a set of “Map” operations that each produces a list of connected pair of sequences as the output. Then these outputs will be sorted and partitioned (if using multiple Reducers) to be processed by the Reducer operation to build a final result set.

\(^2\)http://ark.intel.com/products/85766
\(^3\)http://openmp.org/wp/
\(^4\)http://www.open-mpi.org/
\(^5\)http://hadoop.apache.org/
6.2.3 Extending the Algorithm for Other Types of Hierarchical Clustering

By using the same method for generating pairwise proximities in parallel, this algorithm may be extended to perform other types of hierarchical clustering, such as complete-linkage and average-linkage. However, they involve more challenges in time and space complexity as opposed to single-linkage clustering.

For complete-linkage and average-linkage, the pairwise distance itself needs to be stored instead of only one bit of data as implemented in this project, where 0 represents an unconnected pair of sequences and 1 represents connected sequences. Other than that, the economic search that was described and utilised in this project would not be applicable for complete-linkage nor average-linkage since they do not have the same agglomerative nearest neighbour property or SANN property.

6.2.4 Storing Pairwise Distances as a Cache

Cache is a technique that is widely used in computing to allow the reusing of previous results so that a future request for the same computation may be eliminated. Caching may be implemented for this algorithm by logging the pairwise distances and both IDs of the pair of the sequences into a file as the clustering progresses. This file may be reused in the future to cluster a dataset that share some common amplicons, or to resume an interrupted clustering. During the initialisation, the program would ideally load the cached information by selecting only the pairwise distances which involve amplicons that overlapped with the input dataset. Pairwise distances that were retrieved from the cache may be used to skip some k-mer comparisons and potentially also the expensive pairwise alignments.
Chapter 7

References


Chapter 8

Appendix: Code Listing

Source codes that are included in this appendix are the ones that were considered as the essential part of the project. The complete package of source code may be viewed at https://github.com/mimitantono/p-swarm.

This project is using GNU Affero GPL as the license, which means that it is free to copy, distribute, and/or modify provided that the software will be made available with the same freedom to copy, distribute and/or modify.

8.1 clusterdata.h

```cpp
/*
 * clusterdata Nh
 *
 * created on: mar 1L RP1U
 * author: mimitantono
 */

#ifndef CLUSTERDATA_H_
#define CLUSTERDATA_H_

#include <queue>
#include <vector>

class cluster_data {
    public:
        cluster_data();
        virtual ~cluster_data();
        class scanner scanner;
        int thread_id;
        unsigned long int matches_found;
        unsigned long int qgram_performed;
        unsigned long int scan_performed;
        unsigned long int row_stat;
        unsigned long int * iteration_stat;
        std::vector<unsigned long int> targetampliconids;
        std::queue<unsigned long int> next_step;
        std::queue<unsigned int> next_step_level;
        std::vector<unsigned long int> * next_comparison;
    }
```
# Cluster Data

**Cluster Data:**

- **Created on:** Mar 1, 2015
- **Author:** mimitantono

```cpp
void write_next_comparison(unsigned long int col, unsigned int distance);
void reset();
};
#endif /* CLUSTERDATA_H_ */

## 8.2 clusterdata.cc

```cpp
/*
 * clusterdata.cc
 * Created on: Mar 1, 2015
 * Author: mimitantono
 */

#include "clusterdata.h"
#include "property.h"
#include <string.h>

cluster_data::cluster_data() {
    thread_id = -1;
    next_comparison = new std::vector<unsigned long int>[
        Property::depth + 1];
    matches_found = 0;
    qgram_performed = 0;
    scan_performed = 0;
    row_stat = 0;
    scanner.search_begin();
    iteration_stat = new unsigned long int[Property::depth + 1];
    for (unsigned int i = 0; i <= Property::depth; i++) {
        iteration_stat[i] = 0;
    }
}

cluster_data::~cluster_data() {
    if (next_comparison)
        delete[] next_comparison;
    if (iteration_stat)
        delete[] iteration_stat;
}

void cluster_data::reset() {
    for (unsigned int i = 0; i <= Property::depth; i++) {
        std::vector<unsigned long int>().swap(next_comparison[i]);
    }
}

void cluster_data::write_next_comparison(unsigned long int col_id, unsigned int distance) {
    if (distance <= Property::max_next)
        next_comparison[Property::max_next_map[distance]].push_back(col_id);
}
```

## 8.3 clusterresult.h

```cpp
/*
 * clusterresult.h
 * Created on: Nov 7, 2014
 * Author: mimitantono
 */
```
#ifndef CLUSTERRESULT_H_
#define CLUSTERRESULT_H_

#include <boost/unordered_map.hpp>
#include <string>
#include <vector>

typedef struct cluster_info {
    unsigned long int cluster_id;
    std::vector<unsigned long int> cluster_members;
} cluster_info;

class cluster_result {
public:
    cluster_result();
    virtual ~cluster_result();
    cluster_info * new_cluster(unsigned long int cluster_id);
    long partition_id;
    void merge_cluster(cluster_info* cluster, cluster_info* merge);
    void print(FILE * stream, bool sort);
    void add_member(cluster_info* cluster, unsigned long int id);
    cluster_info * find_member(unsigned long int sequence_id);
private:
    boost::unordered_map<unsigned long int, cluster_info> clusters;
    unsigned long int * member_stat;
};

#endif /* CLUSTERRESULT_H_ */

8.4 clusterresult.cc

/*
 * clusterresult.cc
 * Created on: Nov 7, 2014
 * Author: mimitantono
 */

#include "clusterresult.h"
#include "property.h"
#include <algorithm>
#include "db.h"
#include <string.h>

cluster_result::cluster_result() {
    partition_id = -1;
    member_stat = new unsigned long int[Property::db_data.sequences]);
    memset(member_stat, 0, Property::db_data.sequences * sizeof(unsigned long int));
}

cluster_result::~cluster_result() {
    if (member_stat)
        delete[] member_stat;
}

cluster_info * cluster_result::new_cluster(unsigned long int cluster_id) {
    cluster_info info;
    info.cluster_id = cluster_id;
    clusters[cluster_id] = info;
    return &clusters[cluster_id];
}

struct compare_cluster {
inline bool operator==(const std::pair<cluster_info, std::vector<unsigned long int>> & struct1, 
        const std::pair<cluster_info, std::vector<unsigned long int>> & struct2) {
    return (Property::db_data.get_seqinfo(struct1.second[0])->header < Property::db_data.get_seqinfo( 
            struct2.second[0])->header);
}

struct compare_member {
    inline bool operator==(const unsigned long int id1, unsigned 
            long int id2) {
        return (Property::db_data.get_seqinfo(id1)->header < 
            Property::db_data.get_seqinfo(id2)->header);
    }
};

/**
 * Need to print out consistent format (such as correct result will
 * look exactly the same)
 * this will an expensive method, turn off except for unit test
 * /
void cluster_result::print(FILE * stream, bool sort) {
    long total = 0;
    long clust = 0;
    if (sort) {
        fprintf(stderr, "\nResult will be sorted alphabetically\n"");
        std::vector<std::pair<cluster_info, std::vector<
            unsigned long int>> > > vector_clusters;
    for (boost::unordered_map<unsigned long int, 
            cluster_info>::const_iterator cit = clusters.begin () ; 
        cit != clusters.end(); ++cit) {
        std::pair<cluster_info, std::vector<unsigned 
            long int>> > pair;
        for (unsigned long i = 0; i < cit->second. 
            cluster_members.size(); i++) {
            pair.second.push_back(cit->second. 
                        cluster_members[i]);
        }
        pair.first = cit->second;
        std::sort(pair.second.begin(), pair.second.end () , compare_member());
        vector_clusters.push_back(pair);
    }
    std::sort(vector_clusters.begin(), vector_clusters.end () , compare_cluster());
    for (unsigned int i = 0; i < vector_clusters.size(); i ++ ) {
        for (unsigned int j = 0; j < vector_clusters[i ].second.size(); j++) {
            fprintf(stream, "%s", Property:: 
                db_data.get_seqinfo(vector_clusters 
                [i].second[j])->header);
            total++;
        }
    fprintf(stream, "\n");
    clust++;
    } 
}
```cpp
void cluster_result::merge_cluster(cluster_info* cluster, cluster_info* merge)
{
    for (unsigned long int i = 0; i < merge->cluster_members.size(); i++) {
        add_member(cluster, merge->cluster_members[i]);
    }
    std::vector<unsigned long int>().swap(merge->cluster_members);
    clusters.erase(clusters.find(merge->cluster_id));
}

cluster_info* cluster_result::find_member(unsigned long int sequence_id) {
    if (member_stat[sequence_id] > 0) {
        return &(clusters[member_stat[sequence_id]]);
    }
    return NULL;
}

void cluster_result::add_member(cluster_info* cluster, unsigned long int sequence_id) {
    cluster->cluster_members.push_back(sequence_id);
    member_stat[sequence_id] = cluster->cluster_id;
}
```

### 8.5 cluster.h

```cpp
/*
 * Bigmatrix Nh
 * Created on: Dec 24, 2014
 * Author: mimitantono
 */

#ifndef CLUSTER_H_
#define CLUSTER_H_

#include <pthread.h>
#include "clusterresult.h"
#include "clusterdata.h"

class Cluster {
public:
    Cluster();
    virtual ~Cluster();
    void find_and_add_singletons();
    void print_debug(cluster_data** cluster_data);
    void print_clusters();
    void run_thread(cluster_data* cluster_data, int total_thread);
private:
    pthread_mutex_t row_id_mutex;
    pthread_mutex_t result_mutex;
    cluster_result result;
    unsigned long int current_row_id;
};
```
unsigned long int current_cluster_id;

unsigned long int get_next_row_id();
void add_match_to_cluster(cluster_data * cluster_data, unsigned long int row, unsigned long int col);
void process_row(bool use_reference, cluster_data * cluster_data, unsigned long int row_id, unsigned int iteration);
void qgram_diff_full_row(unsigned long int row_id, cluster_data * cluster_data, bool write_reference);
void walkthrough_row_by_reference(unsigned int iteration, cluster_data * cluster_data, unsigned long int row_id);
void prepare_alignment(unsigned long int col_id, unsigned long int row_id, cluster_data * cluster_data);

};
#endif /* cluster_h_ */

8.6 cluster.cc

/* bigmatrix Ncc */
/* created on: Dec RT L RP1T */
/* author : mimitantono */
*/
#include "cluster.h"
#include "qgram.h"
#include "scan.h"
#include "db.h"
#include "util.h"
#include <locale.h>
#include "seqinfo.h"
#include "property.h"

Cluster::Cluster() {
current_row_id = 1;
current_cluster_id = 1;
pthread_mutex_init(&result_mutex, NULL);
pthread_mutex_init(&row_id_mutex, NULL);
}
Cluster::~Cluster() {
pthread_mutex_destroy(&result_mutex);
pthread_mutex_destroy(&row_id_mutex);
}
unsigned long int Cluster::get_next_row_id() {
unsigned long int return_row = 0;
pthread_mutex_lock(&row_id_mutex);
if (current_row_id <= Property::db_data.sequences) {
    return_row = current_row_id++;
}
pthread_mutex_unlock(&row_id_mutex);
return return_row;
}
void Cluster::run_thread(cluster_data *data, int total_thread) {
if (data->thread_id == 0)
    progress_init("Calculating matrix :", Property::db_data.sequences);
unsigned long int row_id = get_next_row_id();
while (row_id > 0) {
    row_id--; // means that loop should be finished
    if (Property::enable_flag) {
        process_row(true, false, data, row_id, 1);
void Cluster::process_row(bool write_reference, bool use_reference, cluster_data * data, unsigned long int row_id, unsigned int iteration) {
    seqinfo_t * row_sequence = Property::db_data.get_seqinfo(row_id);
    if (row_sequence->is_visited()) {
        return;
    }
    row_sequence->set_visited();
    *data->row_stat = *data->iteration_stat[iteration];
    if (!use_reference) {
        for (unsigned long col_id = row_id + 1; col_id <
            Property::db_data.sequences; ++col_id) {
            seqinfo_t * col_sequence = Property::db_data.get_seqinfo(col_id);
            unsigned long qgramdiff = qgram_diff(
                row_sequence->qgram, col_sequence->qgram);
            if (qgramdiff <= Property::resolution) {
                data->targetampliconids.push_back(col_id);
            } else if (write_reference && qgramdiff <=
                Property::max_next) {
                data->next_comparison[Property::max_next][col_id] = true;
                unsigned long col_id = data->next_comparison[j][k];
                if (col_id > row_id) {
                    seqinfo_t * col_sequence =
                        Property::db_data.get_seqinfo(col_id);
                    unsigned long qgramdiff =
                        qgram_diff(row_sequence->qgram, col_sequence->qgram);
                    if (qgramdiff <= Property::resolution) {
                        data->targetampliconids.push_back(col_id);
                    }
                }
            }
        }
        data->qgram_performed += Property::db_data.sequences -
            row_id - 1;
    } else if (use_reference) {
        for (unsigned int j = 0; j < iteration; ++j) {
            std::vector<unsigned long int> new_comparison;
            for (unsigned int k = 0; k < data->
                next_comparison[j].size(); ++k) {
                bool push = true;
                unsigned long col_id = data->
                    next_comparison[j][k];
                if (col_id > row_id) {
                    seqinfo_t * col_sequence =
                        Property::db_data.get_seqinfo(col_id);
                    unsigned long qgramdiff =
                        qgram_diff(row_sequence->qgram, col_sequence->qgram);
                    if (qgramdiff <= Property::resolution) {
                        data->targetampliconids.push_back(col_id);
                    }
                }
            }
        }
    }
}
push = false;

if (push) {
    new_comparison.push_back(col_id);
}

if (compare_row.col_id == compare_row.col_id) {
    compare_row.new_comparison.swap(compare_row.next_comparison);
}

for (unsigned long j = 0; j < data->targetampliconids.size(); ++j) {
    unsigned long int col_id = data->targetampliconids[j];
    unsigned long int diff = data->scanner.master_result[j];
    if (diff <= Property::resolution) {
        add_match_to_cluster(data, row_id, col_id);
        if (Property::enable_flag & & Property::db_data .get_seqinfo(col_id)->is_visited() & &
            iteration < Property::depth) {
            set_visited();
            data->next_step.push(col_id);
            data->next_step_level.push(iteration + 1);
        }
    } else if (write_reference || use_reference) {
        data->next_comparison[Iteration].push_back(
            col_id);
    }
}

std::vector<unsigned long int>().swap(data->targetampliconids);
if (data->thread_id == 0)
    progress_update(current_row_id);

void Cluster::find_and_add_singletons() {
    for (unsigned long int i = 0; i < Property::db_data.sequences;
        ++i) {
        if (result.find_member(i) == NULL) {
            cluster_info * added = result.new_cluster(
                current_cluster_id++);
            result.add_member(added, i);
        }
    }
}

void Cluster::print_clusters() {
    #ifdef DEBUG
        result.print(Property::outfile, true);
    #else
        result.print(Property::outfile, false);
    #endif
}

void Cluster::print_debug(cluster_data ** data) {
    unsigned int matches_found = 0;
    unsigned int qgram_performed = 0;
    unsigned int scan_performed = 0;
    unsigned int * iteration_stat = new unsigned int[
for (unsigned int j = 0; j < Property::depth; j++) {
    iteration_stat[j] = 0;
}
for (int t = 0; t < Property::threads; t++) {
    fprintf(Property::dbdebug, "Row stat [%d]\t\t: %lld\n",
            t, ((data)[t].row_stat);
    matches_found += (*data)[t].matches_found;
    qgram_performed += (*data)[t].qgram_performed;
    scan_performed += (*data)[t].scan_performed;
    for (unsigned int j = 0; j < Property::depth; j++) {
        iteration_stat[j] += (*data)[t].iteration_stat[j + 1];
    }
}
fprintf(Property::dbdebug, "Total match\t\t: %lld\n", matches_found);
fprintf(Property::dbdebug, "Total estimate\t\t: %lld\n", qgram_performed);
fprintf(Property::dbdebug, "Total search\t\t: %lld\n", scan_performed);
fprintf(stderr, "Total estimate : %ld\n", qgram_performed);
fprintf(stderr, "Total search : %ld\n", scan_performed);
for (unsigned int j = 0; j < Property::depth; j++) {
    fprintf(Property::dbdebug, "Iteration[%d]\t\t: %lld\n", j + 1, iteration_stat[j]);
}
)

void Cluster::add_match_to_cluster(cluster_data * data, unsigned long int first, unsigned long int second) {
    pthread_mutex_lock(&result_mutex);
    cluster_info * existing_first = result.find_member(first);
    cluster_info * existing_second = result.find_member(second);
    if (existing_first != NULL && existing_second != NULL) {
        result.add_member(existing_first, second);
    } else if (existing_first == NULL && existing_second != NULL) {
        result.add_member(existing_second, first);
    } else if (existing_first == NULL && existing_second == NULL) {
        cluster_info * added = result.new_cluster(
            current_cluster_id++);
        result.add_member(added, first);
        result.add_member(added, second);
    } else if (existing_first != NULL && existing_second != NULL) {
        if (existing_first->cluster_id > existing_second->cluster_id) {
            result.merge_cluster(existing_first, existing_second);
        } else {
            fprintf(Property::dbdebug, "Create cluster %ld for %ld and %ld\n",
                current_cluster_id, first, second);
            current_cluster_id++;
            result.add_member(new_cluster, first);
            result.add_member(new_cluster, second);
        }
    } else if (existing_first != NULL && existing_second != NULL) {
        if (existing_first->cluster_id != existing_second->cluster_id) {
            fprintf(Property::dbdebug, "Merge cluster %ld with %ld\n",
                existing_first->cluster_id, existing_second->cluster_id);
        }
        if (existing_first->cluster_members.size() > existing_second->cluster_members.size()) {
            result.merge_cluster(existing_first, existing_second);
        } else {
            fprintf(Property::dbdebug, "Add %ld to cluster %ld\n",
                second, existing_first->cluster_id);
        }
    }
}
result.merge_cluster(existing_second, existing_first);

#endif

#define DEBUG

fprintf(Property::dbdebug, "%ld and %ld are connected\n", first, second);

#endif

++data->matches_found;

pthread_mutex_unlock(&result_mutex);