Fall 12-15-2017

Focus: A Graph Approach for Data-Mining and Domain-Specific Assembly of Next Generation Sequencing Data

Julia Sommer

University of Nebraska Medical Center

Let us know how access to this document benefits you
http://unmc.libwizard.com/DCFeedback

Follow this and additional works at: https://digitalcommons.unmc.edu/etd

Part of the Bioinformatics Commons

Recommended Citation
Sommer, Julia, "Focus: A Graph Approach for Data-Mining and Domain-Specific Assembly of Next Generation Sequencing Data" (2017). Theses & Dissertations. 242.
https://digitalcommons.unmc.edu/etd/242

This Dissertation is brought to you for free and open access by the Graduate Studies at DigitalCommons@UNMC. It has been accepted for inclusion in Theses & Dissertations by an authorized administrator of DigitalCommons@UNMC. For more information, please contact digitalcommons@unmc.edu.
FOCUS: A GRAPH APPROACH FOR DATA-MINING AND DOMAIN-SPECIFIC ASSEMBLY OF NEXT GENERATION SEQUENCING DATA

By

Julia D. Sommer

A DISSERTATION

Presented to the Faculty of
The University of Nebraska Graduate College
In Partial Fulfillment of the Requirements
For the Degree of Doctor of Philosophy

Pathology and Microbiology Graduate Program
Specialty Track in Bioinformatics

Under the Supervision of Professor Hesham H. Ali

University of Nebraska Medical Center
Omaha, Nebraska

December 2017

Supervisory Committee:

Dhundy K. Bastola, Ph.D. Javeed Iqbal, Ph.D.
Dario Ghersi, M.D, Ph.D. Michael R. Green Ph.D
ACKNOWLEDGEMENTS

First I would like to extend my sincere gratitude to my mentor Dr. Hesham H. Ali. I first remember working with Dr. Ali as a freshmen INBRE undergraduate scholar at the University of Nebraska at Omaha. I distinctly remember our first conversation about potential projects and Dr. Ali discussing my options of taking an easier, “low hanging fruit” project or a highly challenging but highly rewarding long-term project. He encouraged me to pursue the challenging but rewarding project. This project led me to discover my passion for science and technology, my decision to pursue graduate school, and the completion of this Ph.D. At that time and continuing to today, I was thrilled that Dr. Ali believed in me and felt that I was capable of handling such a challenging project. Dr. Ali is like this with all of his students, believing in them and bringing out the best of their talents and capabilities. Despite having an extremely busy schedule, Dr. Ali finds time for all of his students and actively goes beyond and above to support them and foster their growth both academically and personally. I thank him for all of his support during both the high times of my academic career as well as through the difficult and challenging times.

I also want to extend my gratitude to all of the members of my graduate and defense committee: Dr. Kiran Bastola, Dr. Dario Ghersi, Dr. Javeed Iqbal, Dr. Michael Green, and Dr. Tammy Kielian. I thank them for their time, invaluable insights and guidance during my Ph.D. It has been a privilege and honor to work with this committee of scientific experts.

Next I want thank others who have shaped and molded my academic career. I give special thanks to Dr. Sachin Pawaskar, with whom I collaborated with during my undergraduate and graduate career. I have thoroughly enjoyed working with Dr. Pawaskar and am a better scientist and computer programmer because of his mentorship and guidance. I thank him for reaching out to me for the next phase of my career as a postdoctoral researcher at the Defense POW/MIA Accounting Agency (DPAA). It has been wonderful to work with those at the DPAA.
with special thanks to Dr. Franklin Damann and other DPAA lab members for warmly welcoming me to the team and teaching me about forensic anthropology.

I am so grateful to have been a part of the UNO Bioinformatics lab and for the camaraderie, encouragement, and support that I have found there. I will always remember all of the conferences traveled to, the tea times attended, and engaging conversations of the UNO bioinformatics lab. I am also thankful for the friendships and support that I found at UNMC during my Ph.D career.

I want to acknowledge the fellowships and scholarships that paved the way for my academic and postgraduate career and the opportunities that they provided. Thank you to Walter and Suzanne Scott Jr. for their generosity through the Walter Scott, Jr. Scholarship, the INBRE program (Grant number P20 RR16469 from the National Center for Research Resources, a component of the National Institutes of Health) for providing undergraduate research opportunities and support, and the UNMC graduate studies fellowship for supporting my graduate research. Thank you also to the Oak Ridge Institute for Science and Education (ORISE) for supporting my research conducted at the DPAA.

I thank my family for all of their love, support, and encouragement. First I wish to thank my wonderful husband and best friend Nathan Sommer for all of his love, humor, and support. He has been the calming force throughout the last stretches of my Ph.D, keeping anxiety and stress at bay through love and laughter. I am looking forward to our next adventures in life! I am forever grateful to my lovely mother Marcia Warnke who was the first to believe in me in all of my endeavors in life whether it was dance, track and field, or academics. I am forever grateful to my brilliant father, Mitch Warnke, who was the first to inspire my interest in science and computer programming. I miss him greatly and wish that he were here to see me complete the PhD. I am thankful to my amazing brother Nick for his support and encouragement and am very proud of his accomplishments in computer science. Thank you also to my special grandmother
Donna Fast for her love and encouragement. I extend my heartfelt gratitude for all of the love and support from my aunts, uncles, cousins, grandparents, in-laws, and friends. Finally, I thank God for being my strength and foundation throughout my life and graduate career.
Next Generation Sequencing (NGS) has emerged as a key technology leading to revolutionary breakthroughs in numerous biomedical research areas. These technologies produce millions to billions of short DNA reads that represent a small fraction of the original target DNA sequence. These short reads contain little information individually but are produced at a high coverage of the original sequence such that many reads overlap. Overlap relationships allow for the reads to be linearly ordered and merged by computational programs called assemblers into long stretches of contiguous sequence called contigs that can be used for research applications. Although the assembly of the reads produced by NGS remains a difficult task, it is the process of extracting useful knowledge from these relatively short sequences that has become one of the most exciting and challenging problems in Bioinformatics.

The assembly of short reads is an aggregative process where critical information is lost as reads are merged into contigs. In addition, the assembly process is treated as a black box, with generic assembler tools that do not adapt to input data set characteristics. Finally, as NGS data throughput continues to increase, there is an increasing need for smart parallel assembler implementations. In this dissertation, a new assembly approach called Focus is proposed. Unlike previous assemblers, Focus relies on a novel hybrid graph constructed from multiple graphs at different levels of granularity to represent the assembly problem, facilitating information capture and dynamic adjustment to input data set characteristics. This work is composed of four specific
aims: 1) The implementation of a robust assembly and analysis tool built on the hybrid graph platform 2) The development and application of graph mining to extract biologically relevant features in NGS data sets 3) The integration of domain specific knowledge to improve the assembly and analysis process. 4) The construction of smart parallel computing approaches, including the application of energy-aware computing for NGS assembly and knowledge integration to improve algorithm performance.

In conclusion, this dissertation presents a complete parallel assembler called Focus that is capable of extracting biologically relevant features directly from its hybrid assembly graph.
# TABLE OF CONTENTS

ACKNOWLEDGEMENTS ................................................................................................. I

ABSTRACT ....................................................................................................................... IV

TABLE OF CONTENTS .................................................................................................... V

LIST OF FIGURES ........................................................................................................... V

LIST OF TABLES ............................................................................................................. VI

LIST OF ABBREVIATIONS ............................................................................................ VIII

CHAPTER 1: INTRODUCTION ...................................................................................... 1
  Problem Statement ........................................................................................................ 2
  Purpose of Research ..................................................................................................... 3
  Research Questions ...................................................................................................... 5
  Significance of Research ............................................................................................. 6
  Organization of Dissertation ....................................................................................... 7

CHAPTER 2: NEXT GENERATION SEQUENCING BACKGROUND ......................... 9
  Next Generation Sequencing: A Brief History ........................................................... 9
  Next Generation Sequencing Technologies ............................................................... 17
    Sequencing by Synthesis: Illumina ........................................................................... 17
    Sequencing by Synthesis: Roche 454 Pyrosequencing ........................................... 19
    Sequencing by Ligation: ABI SOLiD Sequencing .................................................... 21
    Single Molecule Real Time (SMRT) Sequencing: PacBio ....................................... 23
    Nanopore Sequencing: Oxford Nanopore Technologies .......................................... 25
  FASTA/FASTQ File Format ....................................................................................... 27
  Next Generation Sequencing Applications ............................................................... 29
    Whole Genome Sequencing ..................................................................................... 29
    RNA-Seq .................................................................................................................. 31
    Metagenomics ......................................................................................................... 31
    Cancer Genomics ..................................................................................................... 31
    Virology ................................................................................................................... 31

CHAPTER 3: THE ASSEMBLY PROBLEM ................................................................ 33

CHAPTER 4: MAIN TERMINOLOGY AND DEFINITIONS .................................... 37
  General Assembler Pipeline ....................................................................................... 37
  NGS Read Preprocessing ......................................................................................... 37
  Determining Read Overlap Relationships .................................................................. 39
    Suffix Trees ............................................................................................................. 41
    Suffix Arrays .......................................................................................................... 43
    Burrows-Wheeler Transform ................................................................................... 43
    K-mer Enumeration Methods ................................................................................. 46
  Graph Theory: Concepts and Definitions ................................................................. 48
  Graph Theory: Structures and Structural Metrics ..................................................... 54
  Graph Theory: Interval Graph .................................................................................... 59
  Graph Theory for Next Generation Sequence Assembly .......................................... 62
    Overlap-Layout-Consensus Assemblers .................................................................. 62
    String Graph Assemblers ......................................................................................... 68
  De Bruijn Graph Assemblers ...................................................................................... 68
CHAPTER 5: LITERATURE REVIEW

Next Generation Sequencing Preprocessing Tools .................................................. 73
  Fastx Toolkit ................................................................. 73
  Galaxy Toolkit ............................................................... 73
  Trimmomatic Toolkit ....................................................... 74
  PRINSEQ ................................................................. 75

Assembly Tools for Next Generation Sequencing Data ........................................... 75
  De Bruijn Graph Assemblers ............................................. 76
    Euler Assembler .......................................................... 76
    Velvet Algorithm ......................................................... 77
    ALLPATHS Assembler .................................................... 78
    SOAPdenovo .............................................................. 79
  Parallel de Bruijn Graph Assembler ...................................... 80
  ABySS Assembler .......................................................... 80
  Ray Assembler ............................................................. 80
  Iterative de Bruijn Graph Assemblers ..................................... 81
    IDBA Assembler .......................................................... 81
    SPAdes Assembler ........................................................ 82
  Overlap Graph Assemblers .................................................. 83
    CAP3 Assembler .......................................................... 83
    CABOG Assembler ........................................................ 83
    IDBA Assembler .......................................................... 84
    Mira Assembler ........................................................... 84
  String Graph Assemblers ................................................... 85
    SGA Assembler ........................................................... 85
  Metagenomics Assemblers ................................................... 85
    Omega Assembler ........................................................ 85
    MetaVelvet Assembler .................................................... 86
    Ray Meta Assembler ....................................................... 87
    MetaSPAdes Assembler .................................................. 87
  RNA-seq Assemblers .......................................................... 88
    Velvet-Oasis Assembler .................................................. 88
    Trinity Assembler ........................................................ 89
    SOAPdenovo-Trans ....................................................... 89
    TopHat and Cufflinks ..................................................... 90
  Long Read Assemblers ....................................................... 91
    CANU Assembler .......................................................... 91
    HINQE Assembler ........................................................ 91

CHAPTER 6: PROJECT GOALS AND OBJECTIVES ................................................. 93

Problem Statement .................................................................................. 93
Research Purpose ................................................................................... 98
Algorithm Overview ............................................................................... 99
Specific Aims .......................................................................................... 99

CHAPTER 7: FOCUS: A NEW MULTILAYER GRAPH MODEL FOR SHORT READ
ANALYSIS ....................................................................................... 104
Introduction ......................................................................................... 104
Methods ............................................................................................... 106
LIST OF FIGURES

FIGURE 1.1: INNOVATIONS OF THE FOCUS ASSEMBLER ............................................ 4
FIGURE 2.1: CHAIN TERMINATION SEQUENCING ...................................................... 10
FIGURE 2.2: HIERARCHICAL GENOME SEQUENCING ............................................ 13
FIGURE 2.3: ILLUMINA SEQUENCING WORKFLOW .............................................. 20
FIGURE 2.4: ROCHE 454 SEQUENCING WORKFLOW ........................................... 22
FIGURE 2.5: SEQUENCING BY LIGATION ............................................................... 24
FIGURE 2.6: PACBIO SMRT SEQUENCING ............................................................ 26
FIGURE 2.7: NANOPORE SEQUENCING ............................................................... 28
FIGURE 2.8: FASTA AND FASTQ FORMAT ............................................................ 30
FIGURE 3.1: THE ASSEMBLY PROBLEM ............................................................... 34
FIGURE 3.2: ASSEMBLY CHALLENGES ............................................................... 36
FIGURE 4.1: GENERAL ASSEMBLER PIPELINE .................................................. 38
FIGURE 4.2: DOVETAIL AND CONTAINMENT OVERLAPS .................................. 40
FIGURE 4.3: SUFFIX TREE .................................................................................... 42
FIGURE 4.4: SUFFIX ARRAY ................................................................................ 44
FIGURE 4.5: THE SUFFIX ARRAY BURROW-WHEELER TRANSFORM FOR A GIVEN STRING S .......................................................... 45
FIGURE 4.6: AN EXAMPLE OF A GRAPH ............................................................... 50
FIGURE 4.7: AN EXAMPLE OF A DIRECTED GRAPH .......................................... 52
FIGURE 4.8: AN EXAMPLE OF A WEIGHTED DIRECTED GRAPH ......................... 53
FIGURE 4.9: A DISCONNECTED GRAPH ............................................................. 55
FIGURE 4.10: NODE DEGREE ............................................................................. 56
FIGURE 4.11: WALKS, PATHS, AND CYCLES ......................................................... 57
FIGURE 4.12: A CLIQUE OF SIZE FOUR ............................................................... 58
FIGURE 4.13: THE INTERVAL GRAPH ................................................................. 60
FIGURE 4.14: THE TOLERANCE GRAPH ............................................................. 61
FIGURE 4.15: THE INTERVAL GRAPH AND THE OVERLAP GRAPH ................. 61
FIGURE 4.16: THE OVERLAP GRAPH BEFORE AND AFTER TRANSITIVE REDUCTION .......................................................... 64
FIGURE 4.17: TYPES OF GRAPH STRUCTURES DUE TO SEQUENCING ERROR .... 66
FIGURE 4.17: DETERMINING PATHS IN THE SIMPLIFIED OVERLAP GRAPH .... 67
FIGURE 4.19: THE DE BRUIJN GRAPH ............................................................... 70
FIGURE 7.1: FOCUS ASSEMBLY AND ANALYSIS PIPELINE ............................. 108
<table>
<thead>
<tr>
<th>FIGURE</th>
<th>DESCRIPTION</th>
<th>PAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>7.2</td>
<td>Multilayer and Hybrid Graph Sets</td>
<td>108</td>
</tr>
<tr>
<td>7.3</td>
<td>Hybrid Graph Trimming</td>
<td>114</td>
</tr>
<tr>
<td>7.2</td>
<td>Multilayer Graph Set Construction Results</td>
<td>120</td>
</tr>
<tr>
<td>7.4</td>
<td>Selected and Rejected Node Read Classification Error Rates</td>
<td>121</td>
</tr>
<tr>
<td>7.5</td>
<td>Selected Node Counts</td>
<td>122</td>
</tr>
<tr>
<td>8.1</td>
<td>Graph Theoretic Model</td>
<td>129</td>
</tr>
<tr>
<td>8.2</td>
<td>Graph Data Structures</td>
<td>133</td>
</tr>
<tr>
<td>8.3</td>
<td>Graph Coarsening</td>
<td>135</td>
</tr>
<tr>
<td>8.4</td>
<td>Read Classification, Graph Relabeling, and Graph Coarsening Statistics</td>
<td>142</td>
</tr>
<tr>
<td>9.1</td>
<td>The Insertion Sequence and the Composite Transposon</td>
<td>152</td>
</tr>
<tr>
<td>9.2</td>
<td>Shannon’s Index Scores</td>
<td>155</td>
</tr>
<tr>
<td>9.3</td>
<td>Genomic Features and Related Graph Structures</td>
<td>156</td>
</tr>
<tr>
<td>9.4</td>
<td>Taxonomic Read Classification</td>
<td>162</td>
</tr>
<tr>
<td>9.5</td>
<td>Shannon’s Index Score Distribution and Seed Subsystem Assignment</td>
<td>164</td>
</tr>
<tr>
<td>9.6</td>
<td>K-Means Clustering Elbow Plots</td>
<td>169</td>
</tr>
<tr>
<td>9.7</td>
<td>Phylogenetic Clusters of Transposases in the Crohn’s Disease Data Sets with Antibiotic Class Enrichments</td>
<td>171</td>
</tr>
<tr>
<td>9.8</td>
<td>Phylogenetic Clusters of Transposases in the Healthy Disease Data Sets with Antibiotic Class Enrichments</td>
<td>173</td>
</tr>
<tr>
<td>9.9</td>
<td>Transposase Associated Subsystem Differences Between Crohn’s Disease and Healthy Gut Microbiome Samples</td>
<td>175</td>
</tr>
<tr>
<td>9.10</td>
<td>Distribution of Genera for Significant Subsystems</td>
<td>178</td>
</tr>
<tr>
<td>9.11</td>
<td>Level 3 Subsystem Counts for Beta-Glucoside Metabolism, Heme, Hemin Uptake and Utilization Systems, and Maltose and Maltodextrin Utilization</td>
<td>181</td>
</tr>
<tr>
<td>10.1</td>
<td>Graph Theory</td>
<td>187</td>
</tr>
<tr>
<td>10.2</td>
<td>Coverage Depth and Assembly</td>
<td>194</td>
</tr>
<tr>
<td>10.3</td>
<td>Optimal Overlap Lengths</td>
<td>196</td>
</tr>
<tr>
<td>11.1</td>
<td>Greedy Graph Growing</td>
<td>205</td>
</tr>
<tr>
<td>11.2</td>
<td>Kernighan-Lin</td>
<td>208</td>
</tr>
<tr>
<td>11.3</td>
<td>Graph Partitioning Speedup</td>
<td>215</td>
</tr>
<tr>
<td>11.4</td>
<td>Hybrid Graph Set versus Multilayer Graph Set</td>
<td>217</td>
</tr>
<tr>
<td>11.5</td>
<td>Distributed Graph Algorithms</td>
<td>220</td>
</tr>
<tr>
<td>11.6</td>
<td>Distribution of Major Genera Across Partitions</td>
<td>223</td>
</tr>
</tbody>
</table>
FIGURE 12.1: ENERGY AWARE SCHEDULING SYSTEM ................................................................. 228
FIGURE 12.2: PREPROCESSING STEP .................................................................................. 231
FIGURE 12.3: CONTAINMENT CLUSTERING ...................................................................... 232
FIGURE 12.4: PROCESS FLOW DIAGRAM ........................................................................... 234
FIGURE 12.5: EXECUTION DEPENDENCIES OF CONTAINMENT TASKS .............. 235
FIGURE 12.6: EAS-EXECUTION TIME V/S NODES. EXECUTION TIME/OVERHEAD V/S NODES. SPEEDUP CURVE FOR THE ASSEMBLY PROGRAM ............................................. 238
FIGURE 12.7: EAS ENGINE – DYNAMIC NODE ADJUSTMENTS .............................. 241
FIGURE 13.1: FIGURE 1.1 REVISITED – INNOVATIONS OF THE FOCUS ASSEMBLER .................................................................................................................................................. 244
FIGURE 13.2: ASSEMBLY GRAPH TARGET READ ENRICHMENT ............................. 251
LIST OF TABLES

TABLE 7.1: HYBRID GRAPH SET AND MULTILAYER GRAPH SET COMPARISON..... 123
TABLE 7.2: ASSEMBLY COMPARISON........................................................................... 125
TABLE 8.1: SIMULATED DATA SETS............................................................................. 143
TABLE 8.2: BACTERIAL READ DATA SETS .................................................................. 145
TABLE 8.3: READ OVERLAPPING RUNTIME (8 NODES) ............................................. 145
TABLE 8.4: GRAPH COARSENING RUNTIME (SERIAL MERGE-SORT) .................... 145
TABLE 9.1: DATA SET CHARACTERISTICS ................................................................ 160
TABLE 9.2: SEQUENCE FEATURES ON NODES WITH THE HIGHEST SHANNON’S INDEXES .................................................................................................................................. 166
TABLE 9.3: SUBSYSTEMS THAT WERE NOT SIGNIFICANTLY DIFFERENT BETWEEN CROHN’S DISEASE AND HEALTHY MICROBIOME SAMPLES .............................................................. 179
TABLE 10.1: ESCHERICHIA COLI DATA SET CHARACTERISTICS ............................ 192
TABLE 10.2: TOLERANCE ASSEMBLY RESULTS ......................................................... 197
TABLE 11.1: DATA SET CHARACTERISTICS ............................................................. 213
TABLE 11.2: EDGE CUT FOR MULTILEVEL AND HYBRID GRAPH SETS ................. 218
TABLE 11.3: ASSEMBLY STATISTICS ............................................................................ 221
TABLE 12.1: DATA SET CHARACTERISTICS ............................................................. 240
# LIST OF ABBREVIATIONS

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Full Form</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABI</td>
<td>Applied Biosystems</td>
</tr>
<tr>
<td>BAC</td>
<td>Bacterial Artificial Chromosome</td>
</tr>
<tr>
<td>BWT</td>
<td>Burrows-Wheeler Transform</td>
</tr>
<tr>
<td>BWM</td>
<td>Burrows-Wheeler Matrix</td>
</tr>
<tr>
<td>DOE</td>
<td>Department of Energy</td>
</tr>
<tr>
<td>DNA</td>
<td>Deoxyribonucleic acid</td>
</tr>
<tr>
<td>HEM</td>
<td>Heavy Edge Matching</td>
</tr>
<tr>
<td>HGP</td>
<td>Human Genome Project</td>
</tr>
<tr>
<td>NGS</td>
<td>Next Generation Sequencing</td>
</tr>
<tr>
<td>OVL</td>
<td>Overlap</td>
</tr>
<tr>
<td>OTU</td>
<td>Operational Taxonomic Units</td>
</tr>
<tr>
<td>OLC</td>
<td>Overlap-Layout-Consensus</td>
</tr>
<tr>
<td>SA</td>
<td>Suffix Array</td>
</tr>
<tr>
<td>STS</td>
<td>Sequence Tagged Site</td>
</tr>
<tr>
<td>TIGR</td>
<td>The Institute for Genomic Research</td>
</tr>
<tr>
<td>NIH</td>
<td>National Institute of Health</td>
</tr>
<tr>
<td>YAC</td>
<td>Yeast Artificial Chromosome</td>
</tr>
<tr>
<td>ZMW</td>
<td>Zero Mode Waveguides</td>
</tr>
</tbody>
</table>
CHAPTER 1

INTRODUCTION

Since its inception in 2005, Next Generation Sequencing (NGS) has revolutionized the biomedical sciences. Biological sequence data is now being produced at unprecedented rates. This is both a blessing and a curse, as researchers now have potentially unlimited amounts of biological data to work with. However, with this deluge of biological data comes the challenge of processing and transforming this data into usable information. Genomes can range from several thousand to several billion base pairs (bp) in length. Current sequencing technologies can only sequence a few hundred to a few thousand bp at a time depending on technology used. These short sequencing fragments are called reads. While these reads might be small in comparison to the source genome, they are produced at an extremely high coverage of the original sequence. For example if a genome were sequenced at 50x coverage, then theoretically each position in the genome would be coverage by fifty reads if coverage were uniform. The data sets output by next generation sequencing technologies are enormous in size and contain millions to billions of reads in random order. In many projects, these reads must be ordered and merged into a representation of the original target sequence before downstream analysis.

The Assembly Problem:

Given a read data set, utilize overlap relationships between reads to find an ordering of the reads that corresponds to the original target sequence. The optimal result of short read assembly is the complete and error free reconstruction of the original target sequence.
The assembly of millions to billions of short sequencing reads requires computational software tools called assemblers to orient and merge the reads into contigs, which are stretches of contiguous sequence representing consecutive regions in the genome. These assembly tools rely on mathematical models called graphs to support the assembly process.

1.1 Problem Statement

The focus of current assemblers is primarily on assembly; the assembly graph serves primarily as an inert scaffold for the assembly process. However, in contrast we propose that as a structural model the assembly graph is able to capture a wealth of biologically relevant information from the input data set. The assembly process actually loses information as individual reads are merged into contigs; the global relationships between reads and any alternate assemblies are lost. There is so much rich information contained in the assembly graph that is lost as reads are merged into flat contigs.

Second, assemblers that have been developed are very generic in nature and do not take input data characteristics into consideration. Many assembly tools are like a black box, with the input data set having little influence on the assembly process. All assemblers will produce an output given an input data set; however, it is unlikely that one-sized-fits all approaches will produce optimal results in all situations.

Finally, as sequencing throughput continues to increase there is a great need for high performance computing methods to meet researcher needs. Several assembly tools have implemented parallel methods for next generation sequencing assembly. However, parallel methods for next generation sequence are often naïve in nature. These parallel methods are also data agnostic. Further, naïve parallel applications might utilize unnecessary computational resources with minimal gains in computational speed-up times.
1.2 Purpose of Research

The major result of this dissertation is an innovative bioinformatics tool called Focus. Focus is a flexible, graph theoretic model for both analyzing and assembling next generation sequencing data. This assembler relies on a novel graph theoretic platform to provide an innovative representation of read data sets. Unlike previous models that use a single graph to model the assembly graph, Focus utilizes a multiset of graphs across a spectrum of granularity. A single graph is only capable of providing a single view of the read data set; however, a multiset of graphs is able to capture both local and global information at different scales of granularity. In this dissertation, we demonstrate that 1) The novel graph multiset is a powerful data-mining support, allowing for the extraction of biologically relevant features from input read data sets. 2) The Focus assembler is an intelligent, customizable and domain-specific approach that can be tailored to the input data set resulting in better assemblies. 3) Smart, energy-aware parallel computing approaches are developed on the Focus platform, leading to more efficient computational utilization while still meeting researcher deadlines. A high performance computing approach with a distributed assembly graph implemented in the Focus assembler not only improves assembly runtime but also reveals community structure in metagenomics data sets. Figure 1.1 highlights the some of the major differences between the Focus assembler and previous approaches.

In addition to introducing the Focus NGS assembly and analysis program, this dissertation also discusses further developed analytical approaches and case studies in metagenomics, a field stemming from next generation sequencing applied to study microbial environmental communities.
Figure 1.1: Innovations of the Focus assembler in comparison to previous assembly approaches. The top of this figure shows the novel multilevel graph approach of the Focus assembler. The bottom of this figure focuses on parallel computing and the distributed assembly graph of the Focus assembler.
1.3 Research Questions

The purpose of this research is to provide an intelligent, customizable, and domain-specific approach for the assembly and biologically relevant feature extraction from next generation sequencing reads. To achieve this goal, several research questions must be answered.

1) How should the graph theoretic model be developed to represent and support the assembly process?

2) How can this graph model be adapted to facilitate graph-based data-mining of input read data sets?

3) What genomic features present in an input next generation sequencing data set are associated with extractable graph structures?

4) How can assembly and graph parameters be tuned according to domain-specific characteristics of the input read data sets?

5) How can we use data-specific information to improve next generation sequence assembly?

6) What assembly processes can be modified to have a parallel implementation?

7) How can the assembly graph be efficiently distributed across multiple compute nodes for parallel computing?

Additional supporting questions are as follows.

- What methods and data structures will the Focus assembly tool use to handle the massive amounts of read data?

- How will the quality of assemblies produced by Focus and other assemblers be evaluated?
• How will overlap relationships be rapidly and efficiently determined between reads?

• Will reads be trimmed and cleaned by the Focus assembler before assembly?

• Which methods will be used to detect and remove erroneous information from the graph model during assembly?

• What types of sequencing data will the Focus algorithm be developed to analyze and assemble?

1.4 Significance of Research

This research is significant because it provides a novel approach for assembly and analysis of next generation sequencing data. Direct data-mining of sequencing data using the assembly graph is an innovative approach that will allow researchers to discover novel data features in their sequencing data. In this research, graph-mining is shown to be capable of discovering numerous biologically relevant features, such as antibiotic resistance gene distributions in healthy and disease-associated gut microbiomes. Besides data-mining, the Focus assembler is capable of performing assembly as a secondary task, making it a flexible and useful tool.

Further, this Focus assembler tool is an intelligent, customizable, domain-specific approach for the analysis and assembly of next generation sequencing data. Most previous assemblers are black-boxes that do not take any of the input data set characteristics into consideration. This alternate concept of tailoring computational algorithms to their input data is beneficial to the research community as a whole and may lead to better results produced by customized software systems.

The Focus assembler introduces novel approaches for parallel sequence assemble, also built upon the concept of customization to input data characteristics. The Focus assembler utilizes
domain-specific features of the input data set to improve graph partitioning over compute nodes on a high performance computing cluster. We have also developed an energy-aware approach for determining overlap relationships between reads. As the volume of biological data continues to increase, there is a growing need for computational resources. It will become important to the research community to balance limited computational resources with required deadlines.

In conclusion, the Focus assembler presents a data-centric, graph data-mining platform that is also a next generation sequencing read assembly tool. Focus encapsulates parallel computing methods that are intelligent and data-dependent, leading to better performance. The data-dependent distributed assembly graph is shown in this research to reveal community structure in metagenomics data sets.

1.5 Organization of Dissertation

Here the organization of this dissertation as a roadmap for the reader is briefly described. In chapter two, a background survey is presented covering 1) a brief history of next generation sequence and 2) an overview of current sequencing technologies. In chapter three, the problem of next generation sequence assembly is introduced and formally defined. Next, supporting terms, definitions, and theoretical background are discussed in chapter four. In chapter five, a literature review of previous assembly approaches for next generation sequence assembly is given. Following this, the problem statement of the dissertation is discussed along with the four specific aims that this dissertation will address. Chapter six introduces the Focus assembly algorithm with chapter seven describing Focus’s technical implementation details. Chapter seven is mostly algorithmically and computationally focused on the underlying structure and organization of the Focus assembler. The next chapter is devoted to the Focus assembly and analysis tool as a data-mining support. The customization of the Focus assembler according to input data characteristics is then explored using a tolerance graph approach. Addressing the final aim, a parallel, energy-aware implementation of the read overlapping module of the Focus assembler is discussed. A
data-dependent distributed assembly graph for the Focus assembler is also highlighted. Finally, conclusions and future directions are discussed in chapter thirteen.
CHAPTER 2

NEXT GENERATION SEQUENCING BACKGROUND

2.1 Next Generation Sequencing: A Brief History

Next generation sequencing has its roots in Sanger sequencing [1] or chain termination sequencing, as its also called. The classical Sanger sequencing method requires a single stranded DNA sequence, a primer, and DNA polymerase, as well as two types of deoxynucleoside triphosphates (dNTPs). The first type of dNTPs was the usual set consisting of deoxyadenosine triphosphates (dATPs), deoxyguanosine triphosphate (dGTPs), deoxycytidine triphosphate (dCTPs), and deoxythymidine triphosphate (dTTPs). The second type was a modified nucleotide type (ddNTPs) that lacks a 3’ OH group. The lack of this 3’ OH group prevents the formation of a phosphodiester bond between nucleotides and therefore terminates the extension of a growing DNA chain.

The original Sanger sequencing approach was a two-step process. First, the DNA to be sequenced is added to four separate sequencing reactions. Each of the reactions contains all of the standard dNTPs. However, each sequencing reaction also contains one of the ddNTP types at a low concentration. As the DNA polymerase extends the growing DNA chain, ddNTPs are randomly inserted into the DNA chain. After the ddNTP is inserted the growing chain is terminated. The result of the first step of the classical Sanger sequencing method is various DNA chains of all possible lengths with known terminating nucleotides. In the second step of the classical Sanger sequencing method, polyacrylamide gel electrophoresis is used to separate the DNA chains, with each sequencing reaction in a different gel electrophoresis lane. The separated DNA chains can now be read by autoradiography to obtain the sequence of the nucleotides. This produced sequence of nucleotides is called a read. In more modern Sanger sequencing, the ddNTPs are fluorescently or radiography tagged so that automatic sequencing machines can read them. Please see Fig. 2.1 for an illustration of classical Sanger sequencing.
Figure 2.1: Chain termination sequencing (Classical Sanger method). A) The DNA strand to be sequenced is added to four different sequencing reactions containing: a DNA primer, a DNA polymerase, the standard four dNTPs, and a modified ddNTP (either ddATP, ddTTP, ddCTP, or ddGTP). The modified ddNTP is randomly inserted into the growing DNA chains, terminating them at different lengths. B) The terminated DNA chains are separated by gel electrophoresis, with each sequencing reaction added to a separate lane. Once the DNA chains are separated by length and visualized by UV light or autoradiography, the ordering of the nucleotides in the DNA strand can be read.
The first DNA genome to be sequenced using the chain termination method was the bacteriophage ΦX174 by Frederick Sanger and his colleagues in 1977 [1]. To sequence bacteriophage ΦX174, Sanger used restriction enzymes to create random primers from the ΦX174 genome. These primers were used to amplify portions of the ΦX174 genome, which were then merged by their overlapping regions. This method was a forerunner of the shotgun sequencing method, which is introduced later in this section.

Rapidly following the publication of the ΦX174 genome, various other DNA genomes were sequenced. In 1982, Frederick Sanger sequenced the λ bacteriophage [2]. This bacteriophage is 48K nucleotides in length. While small, the genome of this bacteriophage was still much longer than the reads that Sanger sequences was able to produce, which were between 15 and 200 nucleotides in length [1]. During this early era of genome sequencing, two approaches were being heavily debated for sequencing larger genomes [3]. The first approach, which was favored at the time, was the directed sequencing approach. In this approach, the genome to be sequenced was cut into large fragments. These fragments were ordered and used to create a map of the genome being sequenced. The fragments were then cloned followed by sequencing.

The directed sequencing approach evolved over time into hierarchical sequencing. In hierarchical sequencing, the genome to be sequenced was fragmented into pieces and the resulting fragments were inserted into artificial chromosomes. In the earlier days of sequencing Yeast Artificial Chromosomes (YACs) were used; however; these chromosomes were prone to genomic rearrangement and chimerism [4]. The use of YACs was shifted towards the use of Bacterial Artificial Chromosomes (BACs), which resolved many of the issues that researchers were facing. On average a BAC could contain 110 kb of DNA sequencing [5]. Once a library of YAC of BAC clones was established, the next step was to identify the ordering of the clones to form a crude physical map of the genome to be sequenced. This was accomplished by using
fingerprinting to determine overlapping ends of clones [6]. In fingerprinting, a restriction enzyme is used to cut each of the clones. The resulting lengths of the set of fragments produced for each clone become that clone’s fingerprint. Overlapping clones could then be determined by shared fingerprint patterns between clones.

Once the ordering of the YAC or BAC clones was completed and a crude physical map of the genome was established, the large inserts were randomly fragmented again into smaller fragments that are inserted into a vector. In the early days of sequencing the M13 vector was used [7] but additional vector types became available as sequencing technologies advanced. Early M13 vectors only allowed for one end of the insert to be sequenced; however, later plasmids allowed for both ends of the insert to be sequenced to produce mate-pair reads [8]. Libraries of clones were generated for these smaller fragments, which were then sequenced using Sanger sequencing technology.

The second method, which was slowly gaining momentum in the 1980s, was the whole genome shotgun sequencing approach. In this method, the creation of the physical genome map was not conducted. Instead multiple copies of the entire genome were sheared into several thousand or million short fragments. These fragments were then inserted into plasmid vectors, cloned, and then directly sequenced. While, this approach appears simpler than hierarchical genome sequencing, there were many concerns during the early sequencing era for its applicability to larger genomes [3]. It would be over a decade before whole genome shotgun sequencing became the accepted standard approach for sequencing both small and large genomes. Please see Fig. 2.2 for a brief overview of both hierarchical and whole genome shotgun sequencing approaches.

The λ bacteriophage was the first DNA genome to be sequenced using the whole genome shotgun sequencing method. While sequencing this genome, Sanger compared both whole genome shotgun sequencing and directed sequencing.
Figure 2.2: Hierarchical genome sequencing. In this approach, the genome is cut into large overlapping fragments. These fragments are inserted into BACs and a BAC clone library is generated. Fingerprinting is conducted and pair-wise comparisons of the resulting fragment lengths are used to detect overlapping regions between the BAC clones. The BAC clones are ordered according to the overlap relationships to form a physical map of the underlying genome. Further fragmentation of the BAC clones is conducted, a M13 vector clone library is generated and sequenced. B) Whole genome shotgun sequencing. Multiple copies of the entire genome are fragmented into small pieces. The fragments are inserted into plasmid vectors and a plasmid vector clone library is generated. The plasmid vectors are then sequenced.
His comparison led him to conclude that whole genome sequencing was more rapid than directed sequencing approaches [3]. However, one issue that early whole genome sequencing had at the time was that there were gaps in the sequencing. The areas that were not covered by the whole genome shotgun sequencing reads had to be finished using directed sequencing methods. This issue was largely mitigating when mate-paired reads were introduced in 1991 [9]. Mate-paired reads are the product of sequencing both ends of a clone, resulting in a pair of reads with a known distance between them.

The first free living organism to be sequenced was *Haemophilus Influenzae* in 1995 [10]. This was accomplished by J. Craig Venter and his team at the Institute for Genomic Research (TIGR). J. Craig Venter was originally located at the National Institute of Heath (NIH) but left in 1992 to form the nonprofit private research institute TIGR for the purpose of pursuing sequencing projects [11]. Venter and his team applied the whole genome shotgun approach to sequence the 1.83 mb *Haemophilus Influenzae* genome. During this time, the hierarchical sequencing approach was still primarily used because computational methods had not yet been developed to handle the large amounts of data produced by whole genome shotgun sequencing approaches. Whole genome shotgun sequencing was still seen as infeasible for genomes larger than small viruses and bacteriophages. However, J Craig Venter and his team developed novel computational methods that provided the capability to assemble the large number of *Haemophilus Influenzae* shotgun sequencing reads. These computational methods were encapsulated in a software tool called the TIGR assembler [12]. This early assembler was able to assemble roughly 25,000 sequencing read fragments in about 30 hours on a SunSPARC station that had 40 Mbytes of RAM. Venter and his team also introduced the concept of paired-end reads, which made sequencing of larger genomes possible with the whole shotgun sequencing approach. Previously, only single end reads could be obtained from the M13 vectors used in sequencing projects.
Soon after the release of the *Haemophilus Influenzae* genome, several more sequencing projects were undertaken by different research groups. In 1997, the *Escherichia coli* genome was completed [13]. This project had initially begun in 1992 with manual methods and was completed in 1997 as automated sequencing technologies began to become available [14]. The first eukaryote genome to be sequenced was *Saccharomyces cerevisiae* in 1997 [15]. For this sequencing project, a large international consortium was formed with roughly 600 scientists collaborating from North America, Europe, and Japan [14]. Sequencing projects continued to increase in ambition and the sequencing of the first complete animal genome, *Caenorhabditis elegans*, was published in 1998 [16].

As sequencing projects during the early to mid 1990s continued to grow in scale and complexity, sequencing technologies continued to advance and improve. In 1995, Applied Biosystems (ABI) released the ABI PRISM 310 sequencer [17]. This sequencer was introduced capillary electrophoresis, which greatly increased the throughput and automation of the sequencing process. During the early 1990s, fluorescent dye labeling for DNA bases was also introduced, leading to the further automation of Sanger sequencing.

In 1998, Craig Venter and Applera, which was the parent company of ABI, formed the Celera Corporation for genome sequencing. This sequencing company relied heavily on ABI’s new automated sequencing machines [14]. Celera conducted the largest sequencing project up to that time by completing the *Drosophila Melanogaster* genome in 1999 using whole genome sequencing [18]. This was heralded as a great win for whole genome shotgun sequencing; however, there was still considerable debate whether or not shotgun sequencing would be suitable for sequencing of the human genome, a project that was in the midst of development at the time [19].

The Human Genome Project (HGP) began in 1990 and spanned until 2003, when it was formally declared as complete. In 1990, the National Institutes of Health and the Department of
Energy (DOE) formed a memorandum of understanding and setup a five year initial road plan for sequencing the human genome [20]. This was a part of a greater long-term human genome initiative of fifteen years. In the first few years of the Human Genome Project until 1993, David Galas was the leader of the DOE’s genome project and James Watson led the NIH genome project until 1993, when they were succeeded by Aristides Patrinos and Francis Collins respectively. The chosen sequencing strategy for the Human Genome Project was a map-based hierarchical approach. A new five-year plan was established in 1993 to account for rapidly changing sequencing and other biomedical technologies. Goals outlined in this plan included the completion of a 2 to 5-cM genetic map and the completion of a sequence tagged site (STS) physical map at a resolution of 100kb [20]. Other goals included supporting informatics resource development and sequence technology development. The project was anticipated to cost a total of 200 million dollars a year, adjusted for inflation.

Several important landmarks occurred throughout the Human Genome Project. The first detailed map of the human genome was completed in 1994 [21]. Moderate resolution maps for chromosome 3, 11, 12, and 33 as well as a physical map with over 15,000 STS markers were released in 1995 [22][23][24] [25][26].

In 1998, Craig Venter formed Celera Genomics to sequence the human genome, leading to a sequencing race between Celera and the Human Genome Project. The Celera human genome-sequencing project relied on the whole genome sequencing approach in contrast to map based approaches for sequencing the human genome. In 2000, both the Human Genome Project and Celera announced that they had an initial draft of the human genome soon to be released [27] [28]. The completion of the human genome opened the floodgates for numerous other sequencing projects; the next few years after the release of the human genome saw rapid advances in sequencing techniques, computational technologies, and sequencing applications.
In 2005, next generation sequencing technologies were introduced into the field of DNA sequencing [29] [30]. These sequencing technologies greatly surpassed the older Sanger sequencing technologies in both throughput and speed. Next generation sequencing technologies revolutionized nearly all areas biomedical sciences, producing unprecedented massive amounts of biological data. This new type of sequencing technology brought with it numerous challenges such as extremely high coverage, higher error rates, and shorter sequence lengths in comparison to Sanger sequencing technologies. This deluge of data spurred the rapid development of novel computational analysis approaches. The next section describes next generation sequencing technologies in detail.

2.2 Next Generation Sequencing Technologies

Several types of next generation sequencing technologies were developed from 2005 until today. Earlier next generation sequencing technologies include sequencing by synthesis (Illumina), pyrosequencing (Roche 454), and sequencing by ligation (SOLID sequencing). We also discuss newer third generation sequencing technologies including single-molecule real-time sequencing (Pacific Biosciences) and ion semiconductor (Ion Torrent sequencing).

2.2.1 Sequencing by Synthesis: Illumina

Sequencing by synthesis is the foundation of the Illumina next generation sequencing technologies. The Illumina company was the result of the merger of four other biotech companies: Illumina, Solexa (Essex, UK), Manteia Predictive Medicine (Coinsins, Switzerland), and Lynx Therapeutics (Hayward, CA, USA) [31].

According to Illumina, there are four basic steps to its sequencing workflow: library preparation, cluster generation, sequencing, and data analysis [32]. In library prep, the DNA is first purified and then fragmented. Tagmentation is often used as the method for fragmentation [33]. Tagmentation relies on adaptor-loaded transposases to randomly shear the purified input DNA sample and add adaptors to the ends of the generated fragments at the same time. The next
step in library prep is reduced cycle amplification where additional primer binding sequences, indices, and terminal sequences that are complementary to the flow cell oligos are added [34]. Following this step is the cluster generation using bridge amplification. The DNA fragments with added adaptors are loaded into a flow cell which is a glass slide coated with two types of adhered oligos. The adaptors of the DNA fragments are complimentary to the oligo pairs bound to the flow cell. This causes the fragments to form complementary bonds with the first type of oligo. Polymerases then bind to the primer sequences on the DNA fragments and synthesize a complementary strand. The strands are denatured and the original DNA fragment is washed away. The other end of the synthesized DNA strand then hybridizes to the other type of oligo, forming a bridge shape. Polymerases then synthesize the complementary strand of the bridge DNA, to form a double stranded bridge. These strands are then denatured and the bridge amplification process is conducted repeatedly to generate clusters for each original DNA fragment.

After bridge amplification is complete, the reverse strands are cleaved and removed. The 3’ ends of the remaining forward strands are blocked to prevent unwanted priming at that end. Now sequencing by synthesis can begin [35]. In sequencing by synthesis, a primer is extended on the 5’ end. At each extension step a mixture of all four nucleotide types are added to the reaction. These nucleotides are fluorescently tagged and blocked at the 3’ OH to prevent incorporation of more than one nucleotide. Once a nucleotide is incorporated into each growing DNA strand, the remaining free nucleotides are washed away. Imaging is used to detect which nucleotide was incorporated for each cluster. The fluorophores and 3’ OH blocks are cleaved and washed away and the processes is begun again until the sequencing cycles are complete.

Illumina sequencing platforms are capable of producing both single-end and paired end reads. Paired-end reads are obtained by sequencing both ends of a DNA fragment library, producing pairs of forward and reverse reads with a known distance between them. The current
Illumina HiSeq X platform is capable of producing 1.8 Tb of data in a single run (5.3 – 6 billion reads passing filter) [36]. Paired end read lengths can be up to 150 bps in length. Read accuracy is greater than 99.5%, with a tendency towards substitution errors [35]. Please see Fig. 2.3 for an overview of the Illumina sequencing by synthesis approach.

2.2.2 Sequencing by Synthesis: Roche 454 Pyrosequencing

The Roche 454 next generation sequencing technologies rely on sequencing by synthesis as well. The 454 Life Sciences company was founded in 2000 by Jonathan Rothberg in Connecticut. This company specialized in pyrosequencing technologies. Roche acquired 454 Life Sciences in 2007 to bolster its sequencing efforts. After six years, Roche shuttered the 454 sequencing technologies in 2013 [37].

In the 454 pyrosequencing method, DNA is sheared into fragments and an adapter sequence is ligated to the resulting fragments ends [38]. The fragments are then bound via the adapter to beads such that there is one fragment per bead. These beads are then added to an oil emulsion containing a PCR reaction mixture. Once added to the oil emulsion, the fragments are clonally amplified. After clonal amplification is completed the beads are treated to produce single stranded fragments. The beads are added to a fibre-optic slide with pictitre wells on its surface. The pictitre wells are exactly sized to hold one fragment covered bead. Once the beads have been deposited in the pictitre wells, pyrosequencing can begin.

Pyrosequencing relies on a set of four enzymes, DNA polymerase, ATP sulfurylase, luciferase, and apyrase. [38]. Two substrates are added to this reaction as well, adenosine 5’ phosphosulfate (APS) and luciferin. First a DNA primer is added to the single stranded fragments adhered to the beads in the pictitre wells. Cyclically, each of the four dNTPs are added to the reaction individually. The polymerase enzyme extends the primers of the fragments with a complementary base to the current dNTP being added.
Figure 2.3: Illumina sequencing workflow. A) Tagmentation: DNA cleavage and addition of adaptors by transposomes. B) DNA fragments with ligated adaptors. C) Reverse cycle amplification adds additional sequence binding sites and indices. D) Glass slide covered with a lawn of two different adaptors. E) Fragments hybridize with adhered adaptors. F) The complementary strand is synthesized. G) Adaptors on other end of fragment hybridize with second adaptor sequence to form bridges. H) Bridge amplification; bridges are now double stranded. I) Bridges are denatured to form single stranded DNA. J) G-I are repeated numerous times to form clusters. K) Reverse strands are cleaved and washed away. L) Sequencing by synthesis
When the dNTP is added to the growing DNA strand being synthesized by the polymerase an inorganic PPi is released. ATP sulfurylase converts the released PPi into ATP in the presence of APS. The generated ATP provides the energy to luciferase to convert luciferin to oxylucifirein, which produces light that can be detected by a sensor. In this manner, it is possible to detect which beads had fragments to which the current cycled dNTP was added.

The GS FLX Titanium system of Roche, which was launched in 2008, could produce reads up to 700 bps in length with an accuracy of 99.9% [39]. The 454 sequencing technologies had a tendency for generating errors in homopolymer runs. This system produced 0.7 G of data in a single run. Fig. 2.4 displays the 454 pyrosequencing method.

2.2.3 **Sequencing by Ligation: ABI SOLiD Sequencing**

The company Applied Biosystems was founded in 1981 by Sam Eletr and Andre Marion [40]. By 2002, Applied Biosystems had revenues reaching $1.6 billion dollars and was the supplier of one of the first next generation sequencing platforms, the SOLiD system. In 2008, Applied Biosystems merged with the Invitrogen Corporation to for Life Technologies Corporation. Thermo Fisher Scientific acquired Life Technologies Corporation in 2014. For SOLiD sequencing, the input DNA sample is first sheared into a fragment library of the desired size. Adaptors are ligated to the ends of the resulting fragments. Similarly to 454 sequencing technologies, SOLiD sequencing technologies uses emulsion PCR to amplify the fragment library, where each fragment is hybridized to a bead-bound primer [41]. After amplification is complete, the beads are loaded onto a glass slide and covalently attached.

The SOLiD sequencing technologies use an intricate system of octamer probes to determine the ordering of nucleotide bases in a read. The first two nucleotides of the octamer probe are any of the sixteen possible permutations of nucleotides. The next three bases are degenerate universal bases that will hybridize with any of the nucleotides. The final three bases are also degenerate but carry a fluorescent tag.
Figure 2.4: Roche 454 sequencing workflow. A) DNA is sheared and adapters are ligated to the end of the resulting fragments. B) Each fragment is ligated to an adapter attached to a bead. (One fragment per bead.) C) The beads are emulsified in an oil-water mixture containing a PCR reaction mixture. The fragments are clonally amplified. D) Pyrosequencing is used to obtain the nucleotide ordering for each fragment.
This tag can be four different colors; each of the colors corresponds to four of the sixteen dinucleotide permutations in a two-base encoding system. First a primer hybridizes to the universal adapter, which attaches the fragments to the glass bead. The first probe is attached by ligation to the primer. The fluorescent tag is visualized and then the three end nucleotides with the tag are cleaved. The next probe is ligated to the first, visualized, and cleaved. This process is continued for several cycles. There is a fluorescent recording for each dinucleotide at every 5th position. To obtain the fluorescent encoding for the other bases, the primer is offset by one and the ligation process begins again. The primer is offset one nucleotide for a total of five rounds, until a fluorescent tag is determined for each consecutive dinucleotide pair in the read. The first base sequenced by the ligation method is known because it is part of the adaptor sequence. This known base along with the two-base encoding system is used to determine the remainder of the sequence of the read.

The current 5500 W Series Genetic Analyzers are capable of producing up to 320 Gb of data in a single run [42]. Read lengths can reach up to 75 for single-end reads and 50 for mate pair and paired-end reads. SOLiD sequencing by ligation is shown in Fig. 2.5.

### 2.2.4 Single Molecule Real-Time (SMRT) Sequencing: PacBio Sequencing

More recently, third generation sequencing technologies have been introduced to the sequencing market. These technologies are now producing reads that are much longer than reads lengths of previous next generation technologies. PacBio was founded in 2004 in California. This company introduced Single Molecule Real-Time (SMRT) sequencing [43].

SMRT sequencing takes advantage of the inherent properties of the DNA polymerase to obtain long read lengths in a high throughput approach. First, hairpin adaptors are ligated to double DNA stranded templates to form structures called SMRTBells [44]. These SMRTBells are loaded onto a chip called a SMRT cell. These SMRT cells contain numerous nanophotonic visualization chambers called Zero Mode Waveguides (ZMW).
Figure 2.5: Sequencing by ligation. A) Two-base encoding. In the initial sequencing, a fluorescent tag is determined for every fifth dinucleotide. B) To obtain the remainder of the sequence, the octamer probes are offset by one nucleotide for five rounds. At the n-1 position, the last base of the known adaptor is sequenced. Using the two-base encoding, this known base can be used to determine the first base of the read. This second base can then be used to determine the next base. This is continued until the sequence of the read is determined.
A polymerase is tethered to the bottom of the cylindrical ZMW, which then binds to a SMRTBell that drifts into the ZMW. Phospholinked nucleotides, where each of the four nucleotide types has a different fluorescent tag attached to the terminal phosphate, also drift into the ZMZ. During DNA synthesis, the polymerase cleaves away the terminal phosphates and fluorescent tag. While the polymerase is incorporating the nucleotide, the fluorescent tag is excited and emits light that is detected by a sensor. In this manner, the PacBio SMRT sequencing technology is able to determine the nucleotide ordering of a DNA SMRTBell.

The PacBio RS II system can produce between 35,000 and 75,000 reads per run [44]. Read lengths of the PacBio RS II system are 10,000 – 15,000 bp on average. The error rate of PacBio reads is much greater than other previous technologies at a single pass error rate of 13%. Fig. 2.6 provides an overview of the SMRT sequencing technology.

### 2.2.5 Nanopore Sequencing: Oxford Nanopore Technologies

Nanopore sequencing technologies are another third generation sequencing technology. Oxford Nanopore Technologies was founded in 2005 in the U.K. Their first sequencing platform, the MinION, was introduced in 2014 [45]. This company specializes in portable sequencing platforms enabled by nanopore sequencing technologies. The MinION system currently weighs less than 100g and can plug into a laptop for analysis. Oxford Nanopore Technologies also plans to release an even smaller sequencing platform called the SmidgeION, designed to be used with a smartphone.

Nanopore sequencing electrophoretically guides DNA molecules through a protein nanopore [46]. These nanopores are located in an electrically resistant membrane that separates two electrolyte solutions. A small potential is generated across the membrane, which results in a current flowing through the nanopore. A molecule passing through the nanopore produces a characteristic modulation of the electrical current; this current change can be detected and used to identify the molecule.
Figure 2.6: PacBio SMRT sequencing. A) Hairpin adaptors are ligated to double stranded DNA to create SMRTBells. B) These SMRTBells diffuse into wells called Zero Mode Waveguides (ZMWs). These ZMWs are illuminated by a light source below the ZMW, allowing for a detection volume of 20 zeptoliters. The SMRTBells form a complex with a polymerase that is tethered to the bottom of the ZMW. The polymerase begins to synthesize a new DNA from the SMRTBell DNA template. C) Nucleotides labeled with fluorescent tags also drift into the ZMW. As the polymerase incorporates a nucleotide into the growing DNA chain, a light excites the fluorescent tag, which then emits a characteristic light allowing for the identification of the base.
To sequence DNA, hairpin adapters are first ligated to the ends of DNA fragments [47]. Processive enzymes are added to the DNA fragments to form complexes that are then captured by the nanopores. The processive enzyme ensures that the DNA strand passes through the nanopore in a unidirectional manner and one base at a time. The current changes induced by the DNA sequence are read 3-6 nucleotides at a time, called k-mers. In this manner, the entire DNA sequence can be read. The hairpin adapter allows for both of the strands of a double stranded DNA fragment to be read at the same time, producing more accurate results.

The MinION sequencer can produce between 5-10 Gb of sequence data in a single run. Read lengths can reach up to hundreds of kilobases [48]. However, the error rate of the nanopore sequencing technologies still remains high. A recent study found the error rate of the MinION sequencer to be 38.3% [49]. An illustration of nanopore sequencing can be found in Fig. 2.7.

2.3 FASTA/FASTQ File Format

The FASTA/FASTQ plain text formats are commonly used to represent the output of next generation sequencing technologies. This output includes both the sequence read data and its associated quality values, which represent the estimated probability of base-calling error per base. The FASTA format represents the sequence data only, while the FASTQ format represents both the sequence data and quality values.

In FASTQ format, four lines are typically used to describe a sequencing read and its associated quality data. The first line always begins with ‘@’ and contains the description or name of the read. The next line contains the sequence data, typically the IUPAC codes for DNA or RNA. The next line begins with ‘+’ and contains optional information; this line is often times left blank. Finally, the quality values formatted in ASCII characters make up the last line of the FASTQ entry. The FASTA format contains sequence information only. Each sequence entry is limited to two lines. The first line begins with ‘>’ and contains the description or name of the sequence. The next line contains sequence information.
Figure 2.7: Nanopore sequencing. A) A nanopore is inserted into a membrane separating two electrolyte solutions. A potential across the membrane results in a current running through the nanopore. Molecules passing through the nanopore create characteristic disruptions in the current. B) A processive enzyme unzips the double stranded DNA and passes single stranded DNA through the nanopore. Resulting disruptions in the current are read as sequential nucleotide kmers in the DNA sequence. A hair pin adaptor allows for both strands of the DNA to be read at the same time, producing more accurate results.
The first FASTQ formats were used for representing Sanger sequencing reads and their associated quality values [50]. The PHRED quality score system is defined as $Q_{\text{PHRED}} = -10 \times \log_{10}(P_e)$, where $P_e$ is the estimated error probability. In FASTQ files the PHRED scores are encoded as ASCII values 33-126 = $Q_{\text{PHRED}} + 33$.

With the introduction of next generation sequencing technologies, several other variations of quality values were introduced. The Solexa FASTQ format was introduced in 2004. The Solexa quality values are given by $Q_{\text{SOLEXA}} = -10 \times \log_{10} \left( \frac{P_e}{1-P_e} \right)$ and are encoded in FASTQ format as $Q_{\text{SOLEXA}} + 66$. Illumina continued to use the Solexa quality score format for sometime but then switched back to PHRED encoding for its Illumina 1.3+ quality score format. In FASTQ files, the Illumina 1.3+ quality scores are offset by 64. Please see Fig. 2.8 for an example of the FASTQ format.

2.4 Next Generation Sequencing Applications

With the advent of next generation sequencing in 2005 came an explosion of exciting biomedical applications. In this section, we briefly describe several examples of applications of next generation sequencing technologies. The applications of next generation sequencing technologies continue to rapidly expand as they are only limited by the imagination and innovation of research scientists.

2.4.1 Whole Genome Sequencing

Next generation sequencing technologies have allowed for the sequencing of thousands of new genomes. The ability to sequence whole genomes is the foundation that has provided massive amounts of information, revolutionizing all areas of biomedical research such as cancer genomics, infectious disease research, and many more.
FASTQ Format
@read1
ATCGATAGGGATAGCGGGGCTAGAGGAAAAGGGCCC
+read1
IIIIIIIIIIIIIIII...////4AIIIIIIIII;;;FDC2ABDDBBFF@#####

FASTA Format
>read1
ATCGATAGGGATAGCGGGGCTAGAGGAAAAGGGCCC

**Figure 2.8:** A) FASTQ format B) FASTA format
2.4.2 RNA-Seq

RNA-Seq is an approach using next generation sequencing technologies to conduct transcriptome profiling [51]. The transcriptome is the complete set of mRNA expressed from the genome of an organism. The ability to identify mRNA transcripts and their quantity provides important insights into areas such as organism development, disease progression, and organism response to environmental factors.

2.4.3 Metagenomics

Metagenomics is the sequencing and analysis of a community of microorganisms recovered directly from the environment [52]. The ability to sequence a community of organisms at the same time can reveal complex interactions between members, governing community homeostasis and biological function. Metagenomics also allows for the sequencing of microorganisms that are difficult to culture. Examples of biomedically important microorganism communities that have been studied using metagenomics is the human oral and gut microbiome [53][54].

2.4.4 Cancer Genomics

Next generation sequencing technologies have had a major impact on the field of cancer genomes. Paired-end reads have allowed for the discovery of novel genomic rearrangements [55] and fusion genes [56] critical to the evolution of cancer progression. Deep sequencing has led to the identification of mutations specific to cancer subtype, laying the groundwork for a precision genomics approach for cancer treatment [57]. Next generation sequencing continues to lead to new and exciting discoveries in the field of cancer research.

2.4.5 Virology

Many species of viruses consist of numerous subtypes that have varying levels of fitness and virulence. For example, HIV is highly difficult to treat because patients harbor many different
quasispecies that evolve rapidly and develop resistance to antiviral medications. Next generation sequencing has been applied to study the evolution of HIV quasispecies and drug resistance [58]. Next generation sequencing continues to be an important tool for studying the evolution of viruses and other infectious agents.
CHAPTER 3
THE ASSEMBLY PROBLEM

The problem of assembling entire genomes from short sequencing reads has been given many useful analogies. It has been compared to solving a very large and complex jigsaw puzzle [59]. This genome jigsaw puzzle is unlike the typical jigsaw puzzle that contains several hundred pieces or perhaps several thousand pieces (if a more challenging puzzle). The genome jigsaw puzzle contains several million to several billion pieces. Some of these puzzle pieces are damaged (sequencing errors), a large portion look nearly identical to one another (repeats), and finally numerous pieces are missing (sequencing gaps). In some instances, more than one genome jigsaw puzzle might be mixed together. This becomes a very challenging puzzle to solve!

Sequencing technologies produce millions to even billions of short sequencing reads in a single run. Like single jigsaw puzzle pieces, these short reads contain no information within themselves, but are produced at such a high coverage of the original genome that many of the short sequencing reads overlap. These overlap relationships are used to determine an ordering of the sequencing reads that corresponds to the original genomic sequence. These reads are merged to form long stretches of contiguous sequence called contigs. Paired-end read information is used to find an ordering of the contigs for their assembly into a scaffold that can be used for downstream analysis. Figure 3.1 provides a schematic of the assembly problem.

The massive amount of information produced by next generation sequencing technologies prohibits manual analysis, so specialized software tools called assemblers are used to assemble overlapping reads into contigs. Most of these assembly software tools rely on graph theoretic approaches to analyze the reads. Two graph theoretic paths followed in assembly are de Bruijn graph approaches and overlap graph approaches. These two graph theoretic platforms are explored in more detail in chapter four.
**Figure 3.1: The assembly problem.** Multiple copies of a genome or target sequence are sheared. The resulting fragments are sequenced to produce reads at a high coverage of the original target sequence. Many of the sequenced reads overlap due to high sequencing coverage. These overlaps are used to order and merge the reads into contigs, which are shown in red. Paired-end read information is then used to orient contigs to form scaffolds.
In an optimal situation the final goal of a sequencing project is to produce a perfect and complete representation of the target sequence being analyzed. However, next generation sequence assembly remains an extremely challenging task. The massive amount of data produced by next generation sequencing technologies requires computationally intensive methods for analysis and assembly. Due to chemistry limitations, next generation sequencing technologies produce reads that have substitution or insertion and deletion errors. Error rates range between 0.5% for Illumina [35] to 13% for the long PacBio reads [44]. Error rates increase the difficulty of distinguishing true positive overlap relationships between reads from false positive overlaps. Other challenges include features of the underlying genomic region that is being sequenced. Repetitive regions greatly increase the challenge of assembling reads. Read coverage may also be affected by the content of the underlying genomic sequence. For example, read coverage is found to increase with GC content leading to uneven coverage [60]. Uneven read coverage can lead to gaps in the resulting assembly. Other sequencing errors such as chimeric reads may also be present in a read dataset. Chimeric reads are reads that contain sequence from non–adjacent genomic regions; these often affect studies that focus on a specific sequence region such as 16S sequencing [61]. Figure 3.2 illustrates examples of assembly challenges.

As sequencing technologies advance, numerous diverse sequencing applications are being developed, each with its own set of challenges. For example, metagenomics samples are particularly sensitive to introduced contamination as it can be difficult to distinguish contamination from true bacterial species present in the sample [62]. Metagenomics read data sets also have uneven coverage due to the varying abundances of microorganisms in the sample. As another example, RNA-seq also has widely varying sequence coverage due to different gene transcript abundances. Additional challenges in RNA-seq include gene isoform assembly and gene fusion detection. As sequencing technologies and associated challenges continue to evolve, novel algorithms are needed to address researcher needs.
A. Errors and small overlaps

- Is it a sequencing error?
  ATTAGG\textcolor{red}{TAGGTTTGAT}
  TTACATTAG\textcolor{red}{GCCG}

- Is this a large enough fragment overlap?
  ATTAG\textcolor{red}{TTAGTTAGATTAC}
  \textcolor{red}{GGCATTA}

B. Repetitive genomic regions

C. Sequencing gaps

\textbf{Figure 3.2: Assembly challenges.} A) Errors in reads can make it difficult to distinguish between true positive and false positive read overlaps. Small overlaps between reads may be due to repetitive regions rather than a true overlap relationship. B) Repetitive regions in the underlying genome lead to the generation of reads that are highly similar and cannot easily be placed in the final assembly. C) Not all regions of the genome might be adequately sequenced leading to gaps in the final assembly.
CHAPTER 4
MAIN TERMINOLOGY AND DEFINITIONS

In this chapter, we first provide an overview of a general assembler pipeline that might be used for assembly. Following this, we discuss graph theory and graph theoretical frameworks commonly used for various assemblers. Finally, we discuss metrics and approaches for analyzing assembly and assembler quality.

4.1 General Assembler Pipeline

The assembly process typically follows variations of four key steps. First, read data is preprocessed to remove low quality read ends, sequence artifacts, adapters, and other sequence indices. The preprocessing of next generation sequencing reads is highly important and can greatly impact assembler results [63]. Once the reads are preprocessed, they can be assembled with the chosen software tool. In the second step of the assembly process, the assembler determines overlap relationships between reads. In the third step, these overlap relationships are used to create a graph, which is then manipulated and analyzed to determine an ordering of the reads. Finally, using information from the graph as well as long distance information such as read pairs, consensus sequences of contigs and scaffolds are determined. Fig. 4.1 shows a typical assembler pipeline.

4.2 NGS Read Preprocessing

The performance of an assembly tool is dependent on the quality of the input read data set. If a read data set has significant contamination or the read quality is low, the resulting assembly can be highly fragmented and potentially produce incorrect chimeric contigs. Preprocessing to improve read data set quality is usually performed before assembly to improve assembly results. Several packages have been developed for next generation sequencing data set preprocessing.
Read preprocessing

Read overlap detection

Graph model and algorithms

Consensus sequences and scaffolding

Figure 4.1: General assembler pipeline
These packages utilize read quality values to conduct read trimming and filtering or can trim and filter reads based on user input values.

### 4.3 Determining Read Overlap Relationships

Let \( R \) be a set of reads. The read \( r \in R \) can be defined as a string over alphabet \( \Sigma = \{A,T,C,G\} \) with length \( |r| = n \). The substring notation is given by \( r[1, n-1] = r(i), r(i+1), \ldots, r(n-1) \), where \( r(i) \) is the value of \( r \) at position \( i \), \( 0 \leq i < n \). A suffix of read \( r \) is defined as \( r[i, n] \) and the prefix of read \( r \) is defined as \( r[0, i] \).

There are two possible overlapping relationships that a pair of reads, \( r_i \) and \( r_j \), can have with one another. The first is the dovetail overlap, where the suffix of one read is the prefix of the other. The second type of overlap is the containment overlap, where one read is a substring of the other. These overlaps may have indels or gaps in their alignment. An example of the two types of alignments can be found in Fig. 4.2.

To determine the overlap relationship between two biological sequences, a bioinformatics algorithm called the Needleman-Wunsch algorithm [64] (Smith-Waterman [65] for local alignment) is used for alignment. The Needleman-Wunsch algorithm is a type of dynamic programming algorithm that determines the optimal global alignment of two sequences. It takes \( n^2 \) time to align two sequences of length \( n \) with the Needleman-Wunsch algorithm.

Needleman-Wunsh obtains a sequence alignment of two sequences of length \( n \) in three steps. First a score matrix of size \( n^2 \) is initialized. The scores are then calculated and placed into the score matrix. A traceback method is applied in the third step to obtain the alignment from the score matrix. The reader can visit [15] for more details on the Needleman-Wunsh algorithm and its implementation.
Figure 4.2: A) Dovetail overlap B) Containment overlap. Notice that the alignments contain indels and gaps. © 2012 IEEE reprinted with permission
Next generation sequencing data sets contain millions to billions of short sequencing reads. It would be computationally prohibitive to align all sequence reads against all other sequence reads in the data set, an $O(N^2)$ operation, to determine read overlap relationships. To address this issue, assemblers utilize additional data structures such as suffix arrays, suffix trees, Burrows Wheeler transform with the FM index, or k-mer enumeration methods to facilitate the read overlapping process. The first three read overlapping approaches are typically found in overlap graph based assemblers while k-mer enumeration methods are used by de Bruijn graph assemblers to determine relationships between reads.

4.3.1 Suffix Trees

A suffix tree is a computational and mathematical data structure created to rapidly find a pattern $P$ in a large string data set. This text reviews the suffix tree as described by Gusfield in [66]. The strength of the suffix tree is its speed. For a string $S$ of length $m$ it requires $O(m)$ time (linear time) to construct a suffix tree. Once constructed, a pattern $P$ of length $n$ can be searched in $S$ in $O(n)$ time. The suffix tree can be extended to the case where $S$ is a set of strings, such as the case for a read data set. The suffix tree is defined according to Gusfield [66]. Please see definition 4.1 for a graph theoretic definition of a tree.

Definition 4.1 The suffix tree for a string $S$ of length $m$ is a tree that has exactly $m$ leaves, 0,1 ..., $m-1$, which correspond respectively to each suffix $S[0,m-1]$, $S[1,m-1]$, ..., $S[m-1,m-1]$ in $S$. Excluding the root node, each internal node in the suffix tree has at least two children. Each edge of the suffix tree represents a non-empty substring found in $S$. The concatenation of the substrings represented by the edges in the path from the $i$th leaf in the suffix tree to the root forms the suffix $S[i,m-1]$.

Fig. 4.3 displays an example of a suffix tree for the string $ACGTAACGG$. For any given string $S$ for which a suffix tree is built, a terminating character outside the string’s alphabet (in this case the ‘$’ character) is appended to the end of $S$. 
Figure 4.3: A suffix tree. The root and internal nodes are colored black. The leaf nodes corresponding to each suffix in ACGTAACGG are colored blue.
This prevents a prefix of $S$ from being equivalent to a suffix of $S$, which would make construction of a suffix tree for $S$ impossible. The suffix tree requires $O(m \mid \Sigma \mid )$ space for a string $S$ of length $m$ [67].

### 4.3.2 Suffix Arrays

The suffix array is very similar to the suffix tree. The suffix array was first described by Manber and Myers in 1989 [67]. For a given string $S$, a suffix array $SA$ maintains the suffixes of $S$ in ascending lexicographical order. The value $SA[j] = i$ indicates that the ranking of the suffix $S[i, m-1]$ is $j$ in the lexicographical ordering. The advantage of the suffix array is that it requires less space than the suffix tree. For a string $S$ of length $m$, the suffix tree requires $O(m \mid \Sigma \mid )$ of space while the suffix array requires $O(m)$ of space [67]. The suffix array method introduced by Manber and Myers requires $O(m \log(m))$ time in comparison to the $O(m)$ time required by the suffix tree. Using a suffix array, a pattern $P$ can of length $n$ can be searched for in $S$ in $O(n + \log(m))$ time. Please see Fig. 4.4 for an example of the suffix array for the string $ACGTAACGG$.

### 4.3.3 Burrows-Wheeler Transform

The Burrows-Wheeler transform (BWT) is a reversible permutation of an input string $S$. The Burrows-Wheeler transformation was originally introduced for compression [68] but, in combination with the FM index, has become popular for DNA sequence indexing for next generation sequencing alignment applications. The BWT is obtained from the Burrows-Wheeler matrix (BWM), which is constructed by taking all rotations of the input string $S$ and lexicographically sorting them. The BWM of the input string $S = ACGTAACGG$ is given in Fig. 4.5.

The BWT is inherently related to the suffix array. Fig. 4.5 demonstrates how the suffix array is related to the BMT for a given string. The BWT transform for a string $S$ with a suffix array $SA$ is given by the following rules.
<table>
<thead>
<tr>
<th>$i$</th>
<th>$S[i, 9]$</th>
<th>$S[SA[j], 9]$</th>
<th>$SA[j]$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>ACGTAACGG$</td>
<td>$$</td>
<td>9</td>
</tr>
<tr>
<td>1</td>
<td>CGTAACGG$</td>
<td>AACGG$</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>GTAACGG$</td>
<td>ACGG$</td>
<td>5</td>
</tr>
<tr>
<td>3</td>
<td>TAACGG$</td>
<td>ACGTAACGG$</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>AACGG$</td>
<td>CGG$</td>
<td>6</td>
</tr>
<tr>
<td>5</td>
<td>ACGG$</td>
<td>CGTAACGG$</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>CGG$</td>
<td>GG$</td>
<td>7</td>
</tr>
<tr>
<td>7</td>
<td>GG$</td>
<td>G$</td>
<td>8</td>
</tr>
<tr>
<td>8</td>
<td>G$</td>
<td>GTAACGG$</td>
<td>2</td>
</tr>
<tr>
<td>9</td>
<td>$$</td>
<td>TAACGG$</td>
<td>3</td>
</tr>
</tbody>
</table>

**Figure 4.4:** A suffix array $SA$ constructed for string $S = ACGTAACGG$. The lexicographical sorting of the suffixes of $S$ is given by $S[SA[j], 9]$. The suffix array $SA$ contains the values $SA[j] = i$. 
Figure 4.5: The $SA$, $BMT$, $BWT$, and $SA$ for string $S$. The BWT is given by the final column of $BMT(S)$. Notice that the suffix array of $S$ can be obtained by removing all characters after the ‘$’ character in the BMT.
1) $BWT[i] = \{S[SA[i] - 1] \text{ if } SA[i] > 0 \}$

2) $BWT[i] = \{\$ \text{ if } SA[i] = 0\}$

The resulting Burrows-Wheeler transformation is reversible and often results in repeating characters to be grouped in the same region in the BWT string. This makes the BWT string easy to compress by compression algorithms such as Huffman’s encoding or run-length encoding.

The FM index was introduced in 2000 by Ferragina and Manzini [69] as a BWT augmented with auxiliary data structures that allows it to function as a string search index. The FM index consists of the first and last columns of the BMT, which are labeled $F$ and $L$ respectively. In this representation $L$ can be compressed and $F$ can be represented using a single integer per character, making the FM index very space efficient. The FM index makes use of backwards searching, $LF$ mapping, a checkpoint occurrence table, and SA sampling to rapidly locate patterns in a highly space efficient data structure. The FM index has sub linear complexity, in the context of indexed string size, for storage space and can locate a pattern $P$ of length $n$ in $O(n)$ time [69]. These properties of the FM index have made it popular as the foundation for many alignment tools that map short reads to extremely large reference sequences or reference sequence databases [70] [71] [72].

4.3.4 K-mer Enumeration Methods

The k-mer enumeration problem can be stated as the problem of finding the abundances (counts) of substrings of length $k$ in a string or set of strings $S$. Let $S$ be a set of strings. A naïve approach for enumerating all k-mers would be to add each unique k-mer to a hash table and record the number of times that it occurs. However, there is an issue with this naïve approach, namely with speed and memory usage. For a given length $k$ and string $S$, there are $|\Sigma|^k$ different possible k-mers, where $|\Sigma|$ is the alphabet size of $S$. For an alphabet size of 4, as in the case of DNA strings, there would be 16,777,216 different k-mers of length twelve. Processing and
storing k-mers as the length of $k$ increases, becomes exponentially more difficult and computationally prohibitive. Thus several clever approaches have been developed to address the k-mer enumeration problem.

First, it has been observed that there are many unique, low frequency k-mers that are present in next generation sequencing data sets due to sequencing error. For example, the publishers of the panda genome corrected all 27-mers with frequency less than four [73]. In doing so, they were able to reduce their data set size from 8.62 billion unique 27-mers to 2.79 billion unique 27-mers.

Numerous algorithms have been developed for correction of low frequency k-mers due to sequencing error. The majority of current methods rely on the Bloom filter as their primary data structure. The Bloom filter is a probabilistic data structure that can rapidly and efficiently determine whether or not an element is in a set [74]. The tradeoff of this data structure is that, as a probabilistic data structure, it determines whether an element is definitely not in the set or if it might be in the set. Bloom filters are very similar to hash tables, which are data structures with an array and hashing function that computes the index of an element in the array. However, Bloom filters require much less space than hash tables. Bloom filters consist of an array and multiple hash functions. The multiple hash functions are used to set several bits in the Bloom filter array to one, regardless of their previous values. Utilizing multiple hash functions, a Bloom filter can very memory efficiently determine if an element is definitely not in a set or might be in the set. While errors are allowable in the Bloom filter data structure, they still can have a very low false positive rate. Numerous tools such as Musket [75], Bless [76], Bloocoo [77], and Lighter [78] utilize the Bloom filter to efficiently count k-mers.

Other approaches to effectively count k-mers include the use of high performance computing to speed the k-mer counting process. A popular tool called Jellyfish [79] uses a multithreaded hash table approach for quickly enumerating k-mer counts.
4.4 Graph Theory: Concepts and Definitions

The graph is a mathematical data structure that is used to model the relationships between objects in a set. A strong graph model formalizes the data that it represents, lending it structure and organization. Once a well-fitting model is found, correct utilization and manipulation of the graph is key to producing an optimal solution. Graph theory has been foundational for assembly tool development. This section introduces graph theory and reviews basic concepts and definitions.

The graph forms the foundation of the majority of assembly tools. As an informal definition, a graph can be described as a set of vertices or nodes joined by a set of lines or edges. Representing a problem by a graph can provide a different point of view and can make a complex problem much simpler. Graphs can be a powerful tool for solving problems and have been successfully applied in many domains. A small subset of graph theory applications is described as follows.

*Chemistry:* The atoms of a given molecule are represented by vertices and the bonds between atoms are represented as edges in a graph. Graph theory has been applied to study the topology of molecules and different isomeric arrangements among many other applications [80].

*Protein-protein interactions:* Proteins facilitate biochemical processes within the cell. Many proteins accomplish this by interacting with one or more other proteins. These proteins and interactions can be modeled by protein-protein interaction (PPI) networks, where proteins are vertices in the network and interactions correspond to edges between vertices.

*Social structures:* Graphs have been important for modeling relationships in social structures. For example contact networks have been used for disease surveillance [81]. In contact networks, a vertex represents each individual and an edge connecting two vertices indicates that their corresponding individuals have had contact.
Neurology: Graph theory has been an important tool for studying the connections of the brain. Functional or anatomical regions of the brain can be represented by vertexes and connections between regions can be represented by edges. Graph theory has been used to study traumatic brain injury [82], Alzheimer’s disease [83], and many other neurobiology applications.

The concept of graph theory was introduced 1736 with Euler’s solution of the Konigsberg bridge problem. At the time of Euler, there were seven bridges that connected two large islands to each other and to the main land in the city of Konigsberg. The citizen’s of Konigsberg would attempt to cross all seven of the bridges once and only once in a walk of the city. Using graph theory Euler proved that there was no solution to a walk that would allow for each bridge to be crossed once [84].

Definition 4.2 A graph (or network) $G = (V, E)$, where $V$ is a set of vertices (or nodes) and $E$ is a set of edges. Edges define pairwise relationships between vertices.

Definition 4.3 A vertex (or node) is the basic element of the graph. The vertex set (or node set) is denoted by $V(G)$ (or $N(G)$) in this dissertation.

Definition 4.4 An edge is a one or two element subset of $V(G)$. The edge set is usually denoted as $E(G)$ or $E$.

Definition 4.5 Given an edge $e_{ij} \in E(G)$ where $e_{ij} = (v_i, v_j)$, the two endpoint vertices $v_i \in V(G)$ and $v_j \in V(G)$ are adjacent.

Definition 4.6 An edge $e_{ij} \in E(G)$ where $e_{ij} = (v_i, v_j)$ and $v_i \in V(G)$ and $v_j \in V(G)$ is incident to $v_i$ and $v_j$.

An example of a graph can be found in Fig. 4.6.
Figure 4.6: An example of a graph $G$. The vertex set $V$ and the edge set $E$ are listed in the figure.

- $V = \{1,2,3,4,5,6\}$
- $E = \{\{1,2\},\{1,3\},\{2,4\},\{3,4\},\{3,5\},\{4,5\},\{5,6\}\}$
Definition 4.7 A self-loop or loop is an edge $e_{ij} \in E(G) = (v_i, v_i)$ where $v_i \in V(G)$.

Definition 4.8 Multiple edges are at least two edges in a graph $G$ that have the same end vertices.

Definition 4.9 A simple graph $G$ is a graph with no loops and no multiple edges.

Definition 4.10 A multigraph $G$ is a graph that contains multiple edges.

Definition 4.11 A directed graph $G$ is a graph where edges are assigned a direction. Each edge $e \in E(G)$ is an ordered pair of vertices. A directed graph is also commonly called a digraph.

Definition 4.12 Each edge in a weighted graph is assigned a numerical value. This numerical value is called the edge weight.

Fig. 4.7 and 4.8 provide an example of a directed graph and a weighted graph, respectively.
Figure 4.7: An example of a directed graph $G$. This directed graph has a loop and multiple edge.
Figure 4.8: An example of a weighted graph. This graph is also directed.
Graph Theory: Structures and Structural Metrics

Graph structures are used to isolate interesting or important sections of a graph. Structural metrics provide a measurement of a structural property of a graph. Global metrics refer to a whole graph and local metrics refer to a single vertex in a graph. In this section, many important graph theoretic structural properties and structural metrics are defined and described.

Definition 4.13 A graph is **connected** if you can get from any node to any other by following a sequence of edges OR if any two nodes are connected by a path.

Definition 4.14 A directed graph is **strongly connected** there is a directed path from any node to any other node.

Definition 4.15 Every disconnected graph can be split into a number of connected **components**.

Definition 4.16 The **in-degree** is the number of edges entering a node. The **out-degree** is the number of edges exiting a node. The **degree** of a given node equals in-degree + out-degree.

Definition 4.17 A **walk** in a graph G is given by \( v_i e_i v_{i+1} e_{i+1} v_{i+2} \ldots e_{n-1} v_n \), where each \( e_i \) is incident to \( v_i \) and \( v_{i+1} \).

Definition 4.18 A walk in a graph \( G = v_i e_i v_{i+1} e_{i+1} v_{i+2} \ldots e_{n-1} v_n \) is **closed** if \( v_i = v_n \).

Definition 4.19 A **path** in a graph G is a walk \( v_i e_i v_{i+1} e_{i+1} v_{i+2} \ldots e_{n-1} v_n \) with no repeating nodes or edges, except possibly \( v_i \) and \( v_n \).

Definition 4.20 A **cycle** is a closed path, \( v_i e_i v_{i+1} e_{i+1} v_{i+2} \ldots e_{n-1} v_n \), where \( v_i = v_n \).

Definition 4.21 A **clique** is a subset of vertices in an undirected graph G, where each pair of vertices in the subset is adjacent.

Definition 4.22 Let there be three vertices \( v_x, v_y, \) and \( v_z \) and two edges \( e_1 = (v_x, v_y) \) and \( e_2 = (v_y, v_z) \). If there exists an edge \( e_3 = (v_x, v_z) \), this edge is said to be **transitive**.
Figure 4.9: A disconnected graph. Connected components of the a graph are outlined in a red rectangle
Figure 4.10: A directed graph. The in-degrees and out-degrees for vertices 1, 2, and 3 are shown above. The edge (4, 3) is transitive.
Figure 4.11: Walks, paths, and cycles. Node lists of paths, walks, and cycles are shown above.
Figure 4.12: A clique of size four.
4.6 Graph Theory: Interval Graph

In graph theory, there is a special class of graphs called perfect graphs. In many situations, problems that are NP hard (non-deterministic polynomial-time hard) to solve on more general graphs are polynomial on perfect graphs, making it desirable to approach a problem in a way such that it can be represented by a subclass of perfect graphs. One subclass of perfect graphs is called an interval graph. This graph is the intersection graph of intervals on the real line.

The interval graph lends itself as a well-defined theoretical model of the next generation sequencing problem. The reads from next-generation sequencing can be thought of intervals on a linear DNA strand. In the most ideal situation, the overlap graph formed by mapping each read to a node and every overlap relationship to edges would form an interval graph.

The reads from next-generation sequencing can be thought of intervals on a linear DNA strand. In the most ideal situation, the overlap graph formed by mapping each read to a node and every overlap relationship to edges would form an interval graph. However, small or low quality overlap relationships between reads may be due to sequence ambiguity and do not necessarily indicate that the reads occur consecutively in the genome. These types of false positive relationships create incorrect edges in the overlap graph making it unlikely that it will form an interval graph.

To address the issue of false overlaps, a threshold is often used that specifies the minimum size of overlap that must be shared between two reads before an edge is created between the nodes representing them. This added threshold shifts the graph model to a more generalized perfect graph class called tolerance graphs. The interval graph and tolerance graph as a model for next generation sequence assembly is discussed in greater detail in chapter ten. An example of the interval and tolerance graphs can be found in Fig. 4.13 and Fig. 4.14.
Figure 4.13: The interval graph. Nodes represent intervals on a real number line. If two intervals overlap, then their corresponding nodes are connected by an edge in the interval graph.
Figure 4.14: The tolerance graph. Nodes represent intervals on a real number line. If two intervals overlap greater than a given tolerance $t_i$, then their corresponding nodes are connected by an edge in the tolerance graph.
4.7 Graph Theory for Next Generation Sequence Assembly

Graph theory has been foundational for next generation sequence assembly. Traditionally most assemblers follow variations of two graph theoretic approaches. The first is the overlap-layout-consensus assemblers, which are rooted in the theoretical framework of the interval graph. The second type of graph model commonly used in assembly tools is the de Bruijn graph. Both graph theoretic approaches will be discussed in this section. The string graph, which is an extension of the overlap-layout-consensus approach, is also discussed in this section.

4.7.1 Overlap-Layout-Consensus Assemblers

The overlap graph was the first graph to be applied as a model for the assembly process. The overlap graph model for assembly is directly related to the interval graph as discussed in section 4.6. The overlap graph maps to the assembly problem as follows. Each sequencing read is mapped to a node in the overlap graph. Two reads that overlap with each other are linked by edges in the overlap graph. An example of the overlap graph and its relationship to the interval graph can be found in Fig. 4.15.

The overlap relationships between reads are typically found by using one of the string search indexes and alignment algorithms discussed in section 4.3. To avoid false positive edges within the graph, user-defined thresholds for overlap length and overlap identity are used. Let \( r_i \) and \( r_j \) be two reads that share an overlap \( o_{ij} \); this overlap is composed of \( \text{matches} \), \( \text{mismatches} \), and \( \text{gaps} \). The identity of \( o_{ij} \) is given by \( \frac{\text{matches}}{\text{matches} + \text{mismatches} + \text{gaps}} \). The length of the overlap \( o_{ij} \) is given by \( |\text{matches} + \text{mismatches} + \text{gaps}| \).

Once the overlaps between reads are determined, they are loaded into the overlap graph along with the reads. In Fig. 4.16 it is evident that there is redundant information within the overlap graph.
Figure 4.15: The interval graph and the overlap graph. Notice how short sequencing reads can be envisioned as intervals on a line. If the overlap graph was perfect – ie no false positive or false negative edges, it would form an interval graph.
Figure 4.16: The overlap graph before and after transitive reduction. All of the transitive edges have been removed leaving a simplified graph.
For example, there is an edge from node one to node five; however, there is also a path from node one to node five through node three. The edge (1,5) is a transitive edge and redundant information. The majority of overlap graph assemblers prune the overlap graph to remove redundant information such as transitive edges and edges resulting from containment overlaps. Fig. 4.16 shows the original overlap graph and the overlap graph after it has been reduced.

After graph reduction and simplification, overlap graph based assemblers perform error trimming on the reduced graph. Sequencing errors often create characteristic features within an assembly graph. Two types of graph structure are associated with sequencing error. The first is the dead end path. This might occur if the read quality deteriorates towards the 3’ end of the read. The second type of graph structure is the bubble, where two paths diverge and then end at the same node. This type of graph structure can occur when there are sequencing errors in the middle of a read. Please see Fig. 4.17 for an example of common graph structures resulting from sequencing error.

After error trimming on the overlap graph is completed, a simplified overlap graph remains. The problem of finding an ordering of the reads from the overlap graph can be formulated as the Hamiltonian path problem. The Hamiltonian path problem is as follows: given a graph find a path that traverses each node exactly once. For example, in Fig. 4.14 the Hamiltonian path that would reconstruct the original sequence is given by \{1,3,5,2,4\}. While the Hamiltonian path problem may seem simple in its formulation, it is actually a very difficult problem that is NP complete. Most overlap graph based assembler tools utilize heuristic path traversal to find an ordering of reads. The assembler tool extends a potential path as long as possible until an ambiguity is reached, such as a fork in the path. The ordered reads are then merged to form a contiguous stretch of sequence called a contig. Fig. 4.18 demonstrates path traversal in a simplified overlap graph.
Figure 4.17: Types of graphs structures due to sequencing error. The reads $R_1$-$R_5$ are shown above. The read $R_4$ has a low quality end resulting in a dead end path in the graph. In the second group of reads, $R_1$-$R_4$, read $R_2$ has an internal sequencing error. This results in a bubble in the overlap graph.
Initial starting nodes: \( l_1, l_2, l_3 \)

**Figure 4.18: Determining paths in the simplified overlap graph.** Three paths are shown in the above graph. Path 1 is extended until a fork in the graph. Path extension begins again with paths 2 and 3.
## 4.7.2 String Graph Assemblers

The string graph [85] is the simplified overlap graph resulting from removing all transitive edges and edges resulting from a containment overlap. The nodes of this graph are all reads that are not contained within another read. All non-branching paths in the simplified overlap graph are merged into a single composite edge representing the chain of nodes. This simplification greatly reduces the number of nodes in the graph and its complexity.

The challenge with building the string graph from the overlap graph is that it requires $O(N^2)$ time complexity to construct the overlap graph from a set of reads with a total summed length of $N$. To address this issue, assembly paradigms have been introduced that can directly construct the string graph without having to perform pairwise read overlap detection.

In [86] the authors utilized the FM index and Burrow-Wheeler transform to efficiently detect nonreducible overlaps between reads. Using this approach, the time complexity of constructing the string graph was reduced from $O(N^2)$ to $O(N)$. The string graph, which is a variant of the overlap graph, has become popular for numerous other assembly approaches.

## 4.7.3 De Bruijn Graph Assemblers

The second type of graph theoretic model utilized by a large percentage of assemblers is the de Bruijn graph. In 2001, Pevzner et al introduced this alternate graph theoretic model for assembly which was implemented in the Euler assembler [87]. The de Bruijn graph became popular with the introduction of next generation sequencing technologies that produced very short reads at a much greater coverage in contrast to previous Sanger sequencing methods. As with overlap graph assemblers, De Bruijn graph approaches begin by taking a read data set as input. Instead of mapping each read to a node, a de Bruijn graph assembler decomposes each read into its composite set of k-mers. These k-mers become the edges in the De Bruijn graph, while the left and right (k-1)-mer of each k-mer becomes a node in the De Bruijn graph. The k-mers are determined by k-mer enumeration methods as discussed in section 4.3.4. The advantage of the de
Bruijn graph is that no matter how many times a k-mer is present in a read data set it is only ever mapped to a single edge in the graph. This makes the de Bruijn graph agnostic to the read data set size and coverage, an important characteristic when assembling extremely large and high coverage short read data sets.

Similar to the overlap graph assemblers, the de Bruijn graph assemblers rely on a user-input threshold to control the stringency of the assembly process. The choice of k-mer size $k$ greatly influences the ability of the de Bruijn graph assembler to untangle repetitive sequence. A larger choice of $k$ may be able to resolve a greater number of repeats; however, it may also lead to a greater number of gaps in the assembly if the value of $k$ is too large. Finding optimal values for parameters is a critical problem for both overlap graph and de Bruijn graph assemblers.

After the de Bruijn graph is constructed, de Bruijn graph assemblers also trim the de Bruijn graph to remove edges due to erroneous k-mers. As with the overlap graph, graph structures in the de Bruijn graph such as dead-end paths and bubbles are associated with sequencing errors. Assemblers typically remove suspicious features from the de Bruijn graph based on criteria such as coverage and path length.

For the purpose of constructing contiguous genomic sequence from the k-mer spectrum, the shortest sequence that contains every k-mer must be found. Modeling the k-mers using a de Bruijn graph, transforms the shortest substring problem to that of finding an Euler path in the de Bruijn graph. The Euler path problem is the problem of traversing each edge in a given graph exactly once. This problem can be solved in linear time, given by $O(|E|)$ where $|E|$ is the number of edges in the graph [88]. Fig. 4.19 gives an example of a de Bruijn graph constructed for a set of reads.
Figure 4.19: The de Bruijn graph. Reads in a data set are decomposed into their composite k-mers. The k-mers become edges in the de Bruijn graph. The right and left (k-1)-mers become nodes in the graph.
4.8 Assembly Scaffolding

Once contigs are constructed, paired reads are often used to join the generated contigs into larger scaffolds. The scaffolding problem can also be represented graph theoretically [89]. Each contig can be mapped to a node in a scaffolding graph with read mate-pairs as edges linking the contigs. The scaffolding graph can capture 1) orientation of the contigs relative to one another and 2) the approximate distance separating each contig in the scaffold. The orientation of the contigs can be determined because read mate-pairs have a known orientation. Read mate-pairs also have a known distance between the members of the pair. Optimally a scaffolder would produce one scaffold per chromosome.

Numerous scaffolding algorithms exist and many assembly tools include a scaffolding module. Popular independent scaffolding algorithms include Bambus2 [90], MIP [91], and Opera [92]. The scaffolder Bambus2 relies on a greedy approach that hierarchically joins contigs based upon contigs that share the greatest number of mate-pair links. The MIP scaffolder utilizes a scaffolding graph as its primary data structure. In this approach, the MIP scaffolder first maps the mate-paired reads to the contig input data set. The mapped links are represented by a graph. This graph is partitioned into sub-problems that the algorithm solves with mixed integer programming.

4.9 Assembly Quality Metrics

With the high complexity and large size of next generation sequencing data sets, it is challenging to develop metrics to evaluate the quality of assemblies. One of the first quality metrics to be employed was the N50 statistic; it was first used in the 2001 publication of the human genome [93]. The N50 statistic is the length N at which 50% of the assembled sequence is contained in contigs at least the size of N. There has been much discussion on the weakness of the N50 statistic. It has been noted that the N50 statistic can be misleading if the distribution of contig lengths is extremely skewed [94]. The N50 also does not give any indication of the
accuracy of an assembly. The study in [94] found that assembly tools might perform very well when only evaluating assembly quality using one statistic; however, the authors found that assemblers are inconsistent across multiple quality metrics.

To address the shortcomings of the N50 length and to provide better assembly assessment, numerous additional assembly quality statistics have been developed. If reference genomes are available for the assembly, mapping tools can be used to map the contigs back to a reference to determine their accuracy. Assembly coverage of the reference genome can also be calculated by mapping the contigs back to the reference. In [95] the NG50 length was introduced. This statistic is very similar to the N50 statistic except that it measures in terms of the length of the genome being assembled rather than the length of the total assembled sequence. Also in [95] a metric called the block NG50 was introduced that combined the NG50 length with the contig accuracy. Instead of using contig length distribution like the N50/NG50, this metric utilizes maximal alignment block lengths of the contigs aligned to a reference genome. A similar metric called the normalized N50 was introduced in [96].

Several tools have been introduced for assembly quality assessment such as QUAST [97] and REAPR [98]. The QUAST quality assessment pipeline The QUAST assessment pipeline provides contig length statistics such as the number of contigs, total length of contigs, and size of longest contig. It also provides quality statistics such as the number of misassembled contigs. Other statistics include the genome coverage and genome characteristics such as GC content. The REAPR pipeline utilizes coverage depth and read mate pairs to identify misassemblies without a reference sequence.
CHAPTER 5
LITERATURE REVIEW

This chapter focuses on the review of tools and methods previously developed for assembly applications. The organization of this chapter is as follows. First, tools and packages developed for the preprocessing of next generation sequencing reads are discussed. Following this, assembler tools for the assembly process are described. This section is organized by assembler type including de Bruijn assemblers, overlap graph assemblers, and string graph assemblers.

5.1 Next Generation Sequencing Preprocessing Tools

The preprocessing of next generation sequencing data is critical for the performance of downstream tools such as assemblers and read mappers. This section describes several popular read preprocessing and quality control packages.

5.1.1 Fastx Toolkit

This toolkit contains several modules for read trimming and quality processing [99]. Modules include a FASTQ/A trimmer that shortens a read from the 3’ end according to a user-input threshold, a FASTQ/A quality filter that filters reads based on their quality values, and a FASTQ/A clipper that can remove adaptors and linkers. This toolkit can also plot the distribution of the read data set’s quality values and nucleotides for analysis purposes. Modules for manipulating data set format, such as converting FASTQ to FASTA, are also included in the Fastx-toolkit.

5.1.2 Galaxy Toolkit

The Galaxy toolkit contains many modules for FASTQ and FASTA filtering and formatting [100]. The Galaxy toolkit is capable of producing quality statistics with its FASTQ Summary Statistics by column module. This module produces numerous quality statistics per
base position including the minimums, maximums, means, and quartile ranges. These statistics can then be plotted using the boxplot tool for read data set quality analysis. The Galaxy toolkit also contains a read trimmer that can remove bases from either the 5’ or 3’ end of the reads. The reads can be trimmed according to a percentage of their length or by a fixed offset. A FASTQ quality filter included in the Galaxy toolkit allows for the reads to be filtered by their minimum and maximum length and quality scores. The Galaxy toolkit can manipulate the format of FASTQ/FASTA files and paired-end data.

5.1.3 Trimmomatic

Trimmomatic is an Illumina–specific read trimmer. This tool can remove technical sequences such as adaptors and primers from Illumina reads. The user provides the templates for the technical reads. Trimmomatic uses these templates to detect adaptor and primer contamination in two modes, simple mode and palindrome mode. The simple mode finds approximate matches between the user-provided template and read data set using a ‘seed and extend’ approach [101]. This mode can detect technical sequences in any part of the reads and in any orientation as long as there is sufficient overlap. The simple mode has difficulty detecting partial technical sequences at the beginning and ends of reads as there might not be sufficient overlap. Partial technical sequences often occur as a result of ‘adapter read-through’, where the fragment being sequenced is shorter than the read length. Part of the adapter is also sequenced in this case. The palindrome model can detect adapter read-through in paired-end reads. If two reads are the result of paired-end sequencing a fragment that is shorter than the read length, then they will be reverse complements of one another with adaptor contamination on their ends. The palindrome mode uses seed and extend alignment to detect the areas of the paired reads that are reverse complements. If the first part of the paired-end reads are found to be reverse complements, the remainder of the reads is likely adaptor contamination, which is then removed.

Trimmomatic also offers quality trimming based on two methods, a sliding window
method and a maximum likelihood method. The sliding window method scans from the 5’ end to the 3’ end using a window size specified by the user. If the average quality value of the nucleotides in the sliding window drops below a user threshold, then the scanning stops and the remaining 3’ end of the read is trimmed. The maximum likelihood method utilizes a maximum likelihood approach to balance the tradeoff between retaining read length versus retaining errors at the end of the read.

5.1.4 PRINSEQ

PRINSEQ is a popular next generation sequencing read preprocessor developed specifically for metagenomics data [102]. PRINSEQ is capable of filtering reads based on complexity (ex. a low complexity read such as GTGTGTGTG). PRINSEQ uses two approaches for filtering low complexity reads. One is an approach similar to low complexity masking in BLAST searches. The second evaluates word block-entropies using the Shannon-Wiener method to determine regions of reads that are low complexity. PRINSEQ is also capable of predicting adaptors and barcodes at the ends of reads using a k-mer approach. PRINSEQ can also remove sequence duplicates by detecting them using sorting and prefix/suffix matching.

PRINSEQ has quality control capabilities similar to the previous packages discussed in this section. PRINSEQ can provide a summary of statistics such as the read quality value statistics, read length statistics, and GC content distribution. PRINSEQ can filter reads based on complexity, read length, and many other filters. Read trimming functionality is also offered by PRINSEQ. PRINSEQ can also format FASTQ files and can convert FASTQ to FASTA and vice versa.

5.2 Assembly Tools for Next Generation Sequencing Data

In this section, popular assembly tools for next generation sequence assembly are reviewed. This section focuses primarily on graph theoretic approaches for the assembly problem. First, this section describes assemblers by graph theoretic platform type. First de Bruijn graph
assemblers are explored followed by overlap graph and string assemblers. Second, assemblers developed for metagenomics, RNA-seq, and long read assembly are described in detail. Please see chapter four for a theoretical background and description of each graph type.

5.2.1 De Bruijn Assemblers

5.2.1.1 Euler Assembler

The Euler assembler [87] was the first de Bruijn graph approach applied to the assembly problem. The Euler algorithm performs error correction of the input read data set before assembly to minimize tangles and other erroneous graph structures in the de Bruijn graph. The distribution of solid k-mers in the input read data set is used to correct each read. Each k-mer in the read set is labeled as a solid k-mer if it appears in at least M reads.

The corrected reads are used to build the de Bruijn graph. In the de Bruijn graph each k-mer becomes an edge in the de Bruijn graph. The right and left (k-1)-mers become the nodes in the de Bruijn graph. The problem of constructing the contigs becomes the problem of finding a path that contains all of the edges in the graph. This is equivalent to the Chinese Postman Problem, which is closely related to the Euler Path problem of traversing each edge of a graph exactly once.

The addition of reads to the de Bruijn graph, which cover paths of k-mers in the graph, transforms the Eulerian Path Problem into the Eulerian Superpath Problem. The Eulerian Superpath Problem is as follows: given a set of paths in a graph, find a Eulerian path that contains the set of paths as subpaths. The presence of errors in the input read data set and repeats in the underlying genome introduces tangles into the de Bruijn graph. These tangles are paths that begin and/or end with a node that has an indegree greater than one and/or and outdegree greater than one, respectively. The Euler assembler utilizes reads to determine which k-mers belong in the same path to untangle resolvable tangles in the de Bruijn graph.

The Eulerian Superpath approach is used to generate contigs from the de Bruijn graph.
that are consistent with the read sequences in the input data set.

5.2.1.2 **Velvet Algorithms**

The Velvet algorithms were developed in 2008 for the assembly of short read sequences [103]. The Velvet assembler models the assembly problem using the De Bruijn graph. The construction of the de Bruijn graph is controlled by the k-mer size parameter provided by Velvet. Smaller k values lead to a greater chance of observing false-positive overlaps between reads, while larger k values lead to fewer false-positives but reduced connectivity in the de Bruijn graph. Thus, it is essential to find the correct balance of k-mer specificity and sensitivity for the purpose of producing an optimal assembly.

After the de Bruijn graph is constructed, Velvet utilizes a series of error correction algorithms for the purpose of simplifying the de Bruijn graph. Velvet concentrates on topological features of the de Bruijn graph to detect errors. Errors create three different distinguishable features in the de Bruijn graph including: tips, bulges, and erroneous connections.

A tip is a short chain of nodes that leads to a dead end path in the de Bruijn graph. Tips are caused by errors at the end of the reads. Velvet removes these tips by removing all dead end paths that are shorter then 2k, where k is the value of the k-mer parameter. The authors’ rational for choosing the values of 2k is because this length is greater than the length in k-mers of an individual short read. The authors reason that errors that involve the entire length of a read are highly unlikely.

The second type of topological feature that Velvet removes from the de Bruijn graph is the bulge. A bulge in the de Bruijn graph is due to sequencing errors in the center of a read. Velvet utilizes a Dijkstra-like breadth-first search to detect and remove redundant paths in the de Bruijn graph.

Finally, Velvet removes erroneous connections. Unlike tips and bulges, erroneous connections do not create any recognizable feature in the de Bruijn graph. Instead, Velvet
employs a coverage cut off metric to determine false-positive edges. After error correction is complete, contig information is extracted from the de Bruijn graph.

The major bottleneck in the Velvet algorithms is the construction of the de Bruijn graph. The authors’ assembly of the Streptococcus genome required 2.0 Gb of ram to construct the de Bruijn graph. The Velvet algorithm tends to become memory hungry, making it suitable only for bacterial sized genomes.

5.2.1.3 **ALLPATHS Assembler**

The ALLPATHS assembler [104] is a de Bruijn graph based assembler. This assembler uses read pairs for local assembly in the de Bruijn graph. The ALLPATHS assembler determines “all paths” between read pairs and then uses this information for local assembly.

Following the approach by the Euler assembler, the ALLPATHS assembler first corrects reads based on the solid set of k-mers in the read input data set. As before in Euler, a k-mer is considered solid if it occurs more than $M$ times in the read data set. These solid k-mers are used to correct errors in the reads.

The first step after read correction in the ALLPATHS assembler pipeline is to determine unipaths in the de Bruijn graph. A unipath is defined as a maximal unbranching path in the graph. Once the unipaths have been determined the ALLPATHS assembler selects a subset to use as seeds for local assembly. The seed unipaths are chosen based on their coverage (unipaths estimated to be a part of a repeat are not preferred as seeds) and length (longer unipaths are preferred). Read pairs are used to select seeds that are spaced apart such that the resulting local assemblies will overlap with one another by a few kilobases.

The ALLPATHS assembler then assembles the neighborhoods around the seed unipaths. A neighborhood is defined by the authors as the unipath seed sequence and the 10 kb of genomic sequence to the right and left of the unipath seed. Reads that map to non-repetitive unipaths in the neighborhood as well as their read pairs are classified as the “primary read cloud”. Reads and
their corresponding pairs that can be assembled from the primary read cloud are classified as the “secondary read cloud”. These read pairs are progressively merged together to form longer reads that have a known estimated separation between them.

Once these large read pairs are obtained, the ALLPATHS algorithm determines the closure between the individual right and left reads of the long read pairs. The closure is defined as a sequence of reads that covers the genomic region between a read pair. The ALLPATHS assembler determines closure by using depth-first search to extend ALL read PATHS between the read pairs. A read path is extended by finding a minimal read extensions—i.e., a read that extends the current read path by a minimal amount.

Once closure is complete, the closed read pairs are glued together to form a local assembly graph for the neighborhood. These local assembly graphs are glued together to form the global sequence graph. Reads and read pairs are mapped back to the final assembly graph. The graph is corrected for errors and ambiguity and the final contig assembly is generated.

5.2.1.4 SOAPdenovo

The SOAPdenovo assembler [105] is a de Bruijn graph based assembler that follows steps typical to de Bruijn graph assembly. First, k-mer frequencies are used to detect and correct errors in the read data set. Reads are scanned to detect regions with low frequency k-mers. For each low frequency k-mer detected, the assembler attempts to change the low frequency k-mer to a higher frequency k-mer if one meeting frequency criteria is found.

After error correction, the de Bruijn graph is constructed. For the SOAPdenovo assembler each k-mer becomes a node in the de Bruijn graph with nodes connected by edges if their respective k-mers overlap by k-1 nucleotides. The constructed graph is then error corrected with dead-end tips shorter than 2k removed from the de Bruijn graph. Small repeats that are shorter than read length are resolved with reads. Bubbles in the de Bruijn graph are detected with the Dijkstra’s algorithm and are merged if the two component paths of the bubble meet similarity
thresholds.

Once graph correction is complete, paired reads are mapped back onto the contig sequences obtained from the graph. The paired-end read mapping is used to construct a linkage graph of the contigs. Repetitive contigs are masked in this graph and transitive linkages are removed. Unambiguous linear paths in the linkage graph become the final scaffolds of the assembly.

5.2.2 Parallel de Bruijn Graph Assemblers

5.2.2.1 ABySS Assembler

The parallel ABySS assembler was developed to overcome the challenges of assembling large eukaryote genomes [106]. Like the Velvet assembler, ABySS relies on the de Bruijn graph as its primary data structure. To address the enormous memory requirements of the de Bruijn graph, ABySS implements a distributed version of the de Bruijn graph.

To begin building the de Bruijn graph, ABySS generates all possible k-mers in the sequence reads. ABySS distributes each k-mer across a cluster of nodes using a sequence based hashing algorithm. For the purpose of storing adjacency information, a compact edge representation was developed. Each k-mer vertex can have up to eight edges, representing a one-base extension in either direction. These edges can be represented using a single byte, where each bit can be flagged to represent the presence or absence of an edge.

ABySS applies similar error correction algorithms as Velvet to simplify the de Bruijn graph. Using both contig merging and paired read information, ABySS extracts contig information from the distributed de Bruijn graph.

5.2.2.2 Ray Assembler

The Ray assembler [107] is a de Bruijn graph based assembler that is capable of assembling mixtures of reads obtained from different sequencing technologies. The advantage of using reads from different sequencing technologies is that the strengths of one read type
compensate for the weaknesses of the other read type. For example, longer read technologies such as PacBio and Nanopore have a much greater error rate (10-20%) than shorter read technologies such as Illumina. Long reads can be combined with short reads to make up for the long reads’ error rate and the short reads’ length. However, it is challenging for assembly tools to integrate reads from different sequencing sources.

This tool relies on the de Bruijn graph as its primary data structure. The Ray assembler calculates the coverage for each k-mer in the input read data set. Each k-mer is considered c-confident if it appears in the read data set at least c times. Only k-mers that are c confident are mapped to nodes in the de Bruijn graph. If a k+1-mer exists in the reads and it is (c-1)-confident, then it is mapped to an edge in the de Bruijn graph. The de Bruijn graph is then annotated with reads. If the first c-confident k-mer in a read, occurs more than 255 times, then that read is assumed to be from a repetitive region and is excluded from the annotation. Each c-confident k-mer node is annotated with the reads such that it is the first c-confident k-mer in that read. A seed an extension algorithm is used to extract paths from the de Bruijn graph for contig construction. An extension of a seed is terminated, if the direction of the extension from the graph is not clear. Similar to Abyss, the Ray assembler uses message passing to distribute data across multiple processors.

5.2.2 Iterative de Bruijn Graph Assemblers

5.2.2.1 IDBA Assembler

The IDBA assembler [108] was developed for de novo and metagenomics assembly. This assembler relies on the de Bruijn graph model for assembly. The de Bruijn graph relies on the parameter k to determine the k-mer length. The selected value of k is critically important for the quality of the resulting assembly. If k is too small, there will be an increased number of false positive edges in the de Bruijn graph. If k is too large, there will be an increased number of gaps in the assembly.
To address the issue of the selection of \( k \), the IDBA assembler iterates through multiple values of \( k \) from \( k_{\text{min}} \) to \( k_{\text{max}} \) and maintains an accumulated de Bruijn graph. In the first iteration the graph \( G_k \) is constructed with \( k_{\text{min}} \) as the initial \( k \)-mer length. This graph is then trimmed and corrected using typical assembly correction algorithms. The graph \( G_k \) is converted into \( G_{k+1} \) by converting the edges of the de Bruijn graph into nodes and connecting nodes with an edge if a \((k+2)\)-mer exists in the reads. The IDBA assembler is also capable of using a user-input step size of \( s \) to generate the next graph level. By using an iterative approach, the IDBA assembler is able to overcome some of the challenges of choosing the optimal value of \( k \) for assembly.

5.2.2.2 SPAdes Assembler

The SPAdes assembler [109] was originally developed for the purpose of assembling single cell sequencing data with very uneven coverage. Similar to the IDBA assembler, SPAdes iterates through multiple \( k \)-mers to construct a “multisized de Bruijn graph”. Before assembly the SPAdes assembler employs an error correction and trimming, read-preprocessing algorithm that is modification of Hammer[110].

After error correction of reads, the SPAdes assembler constructs the multisized de Bruijn graph with a range of \( k \)-mer values. The SPAdes assembler performs graph error correction on the multisized graph with the correction and removal of bulges and tips from the graph. SPAdes then incorporates read pair information into the multisized graph to create a paired de Bruijn graph. Unlike most assembler, SPAdes breaks down the paired reads into \( k \)-bimers or pairs of \( k \)-mers. SPAdes utilizes a series of steps (\( B, H, E, \) and \( A \) transformations) to obtain an exact estimate of the distance between \( k \)-mer pairs in the de Bruijn graph. These four steps involve transforming paired reads into paired \( k \)-mers (\( k \)-bimers). The paired \( k \)-bimers are used to estimate genomic distances of paths in the de Bruijn graph, which are loaded into a histogram. This histogram of estimated distances is then used to obtain an exact estimate of distances between \( k \)-bimers. The paired de Bruijn graph is constructed using information from the \( k \)-bimer
adjustment. Finally contigs are obtained from the paired de Bruijn graph and reads are mapped to
the contigs by backtracking graph simplification steps.

5.2.3 Overlap Graph Assemblers

5.2.3.1 CAP3 Assembler

The Cap3 algorithm was developed originally for the assembly of Sanger reads, but is still used today for 454 assembly [111]. The Cap3 assembly program begins by identifying and clipping low quality 5' and 3' read ends. After the reads are cleaned, Cap3 identifies potential overlaps between fragments using a BLAST-like technique. A banded Smith-Waterman algorithm that is weighted by quality values is used to confirm high confidence overlaps. Each detected overlap is evaluated by five criterion: minimum overlap, percent identity, similarity score, differences of bases with high quality values, and the difference rate of the overlap.

Once all of the read overlaps are detected and confirmed, Cap3 uses a greedy method to establish a layout of the contigs. This greedy method operates by constructing the contigs in the order of decreasing overlap scores. The read layout is checked for read pair constraint consistency. If there is a region of the read layout where multiple read pair constraints are not satisfied, then that area is reevaluated by the algorithm. The final steps of the algorithm order the contigs using read pair constraints and derive the contig consensus sequences using weighted multiple alignment.

5.2.3.2 CABOG Assembler

The CABOG assembler is a revised pipeline of the Celera assembler, specifically developed for the assembly of 454 reads [112]. This pipeline is made up of modules for different phases of the assembly process. First, the CABOG assembler begins with read error correction with its overlap-based trimming module. This trimmer computes overlaps for all read pairs. The trimmer flags regions of reads that have discontinuous overlap coverage and trim the reads until it reaches an overlap confirmed span.
CABOG then uses a seed-based overlap anchoring algorithm that is robust to the homopolymer errors that are commonly present in 454 reads. The algorithm begins by compressing the reads such that runs of the same base are reduced to a single base. The algorithm then tabulates the number of occurrences of each k-mer in the reads. To avoid using k-mers that originate from a repetitive region, the algorithm imposes a threshold $M$ such that k-mers that occur more than $M$ times are not used in the overlap seeding process. In the alignment process, each read is considered to be an anchor to which all other reads are aligned. This multi-alignment is used to detect and correct sequencing errors.

The next phase of the CABOG assembler loads the reads into a specialized overlap graph called the best overlap graph. When the CABOG assembler builds the best-overlap graph, it disregards overlaps from containment fragments. It also only allows for one directed edge per node. This edge corresponds to the node's corresponding read's best overlap. The best overlap is defined to be the read’s longest overlap.

CABOG scores each read according to how many other reads are reachable from it. It sorts the reads in descending order by score. CABOG then constructs contigs starting with the highest scoring reads and traversing the unique paths in the best-overlap graph.

5.2.3.3 Mira Assembler

The Mira assembler was developed with special emphasis on the assembly of repetitive sequence [113]. The Mira algorithm begins by comparing each read to every other read using a fast shift pattern matching algorithm for the purpose of detecting potential overlaps. The next step of the algorithm confirms the potential overlaps using a modified banded Smith-Waterman alignment. The confirmed overlaps are loaded into an overlap graph.

Mira employs a semi-greedy algorithm called pathfinder to traverse the overlap graph and construct contigs. Pathfinder begins at the nodes with the heaviest weighted edges. Rather than greedily selecting the heaviest weighted edge to extend the contig, Pathfinder uses a recursive
look ahead method to determine the heaviest weighted partial path. The algorithm then selects the next edge from the heaviest weighted partial path.

5.2.4 **String Graph Assemblers**

5.2.4.1 **SGA Assembler**

The SGA assembler (String Graph Assembler) [114] utilizes the FM index to efficiently construct a string graph for assembly. First the FM index is constructed for the reads and used for error correction. Following read correction, duplicate and exactly contained reads are removed from the corrected read set.

Once read correction is complete, the SGA assembler begins to build the string graph from irreducible or non-transitive overlaps. The SGA assembler locally constructs a string graph around each read by querying the FM index for non-transitive overlaps. A maximal non-branching path is extended from the initial seed read. Once path extension is complete, the reads in the path are collapsed into a single node. After the graph is constructed, the SGA assembler performs typical graph trimming and error correction processes. In the final step, the assembler scaffolds the contigs using paired-end or mate-pair reads.

5.2.5 **Metagenomics Assemblers**

5.2.5.1 **Omega Assembler**

The Omega assembler [115] is an overlap graph based assembler developed specifically for metagenomics data. Omega uses a hash table to index the input read data set. The prefixes and suffixes of the reads are keys in the hash table. This hash table is used to rapidly find exact match overlap relationships between reads.

Once the overlap relationships have been determined, the reads are loaded as nodes in the overlap graph. Bi-directed edges are used to represent exact overlaps shared between reads. Edges have direction depending on whether one or more of the overlapping reads is a reverse complement or not. Once the graph is constructed all of the transitive edges are removed and linear non-branching paths are contracted into a single edge. Graph cleaning removes small dead-
end paths and bubbles from the overlap graph.

For the purpose of determining repeat content in the assembly, minimum cost flow analysis is applied to the error corrected overlap graph. After minimum flow analysis is complete, edges with a unit flow greater one are classified as a repetitive in a single genome or as a shared region in multiple genomes. Edges that have a final flow value near to zero are trimmed from the graph.

Paired-end reads are used to merge adjacent composite edges together. Two composite edges are merged into a single edge if they are connected by three or more paired reads. After merging composite edges supported by paired reads, the paired-end reads are used to scaffold non-adjacent edges.

In the last step before contig construction, the Omega assembler attempts to resolve ambiguous nodes using coverage depth. The authors define an ambiguous node as a node with multiple in and out edges. Coverage depth is used to match an in-edge to an out-edge if possible. After ambiguity resolution, the Omega assembler generates the contigs and scaffold sequences.

5.2.5.2 MetaVelvet Assembler

MetaVelvet [116] is an extension of Velvet for the assembly of metagenomics data. In this paper, the authors note that individual species in a metagenomics mixture might correspond to subgraphs in the overall de Bruijn graph structure. The authors decompose the original mixed de Bruijn graph into individual subgraphs that are likely to correspond to individual genomes.

The MetaVelvet assembler separates subgraphs based on coverage. Two species in a metagenomics data set might be present at different abundances. Thus if two subgraphs have different k-mer coverages, it would be reasonable to assume that they represent two different genomes. The MetaVelvet assembler identifies shared nodes between subgraphs and disconnects them using paired-read information.
Once the individual subgraphs are obtained, the MetaVelvet assembler applies the standard Velvet assembler algorithms to assemble each subgraph.

5.2.5.3 Ray Meta Assembler

The Ray Meta [117] assembler is an extension of the parallel Ray assembler for metagenomics data. The Ray Meta differs from the original Ray assembler in several ways. The Ray assembler uses the global average coverage of the k-mers to select seeds for extension into contigs. In Ray Meta coverage values are sampled in local regions of the graph. This is to account for the differing abundance levels between genomes in a metagenomics data set.

Once the graph construction and editing is complete, the Ray Meta assembler uses a heuristics-guided graph transversal algorithm to generate assembled contigs. The authors of Ray Meta also introduced a taxonomic profiling tool included with Ray Meta called Ray communities. This taxonomic profiler is used to color the Ray de Bruijn graph according to the distribution of metagenomics taxa.

5.2.5.4 MetaSPAdes Assembler

The metaSPAdes [118] assembler is an extension of the SPAdes assembler developed for metagenomics assembly. The metaSPAdes assembler does not attempt to assemble each individual strain present in a metagenomics read data set. The goal of this assembler is to construct a consensus sequence for closely related strains present in the data set.

While constructing the strain consensus sequence, the metaSPAdes assembler uses modified error correcting procedures from SPAdes, such as tip and bubble removal, to mask strain variation in the assembly graph. The authors label an edge resulting from a rare strain as a filigree edge. Coverage ratios between edges are used to identify these filigree edges. Edges identified as filigree edges are disconnected from but not removed from the de Bruijn graph.

A SPAdes module called exSPAder utilizes paired-end read and/or long read information to detangle repetitive regions in the de Bruijn graph. MetaSPAdes also uses read coverage
information to distinguish between intra-species repeats if read pair information is not sufficient.

The exSPAdes module is used to generate strain-contigs and strain-paths that represent individual strains. These strain-contigs are used to resolve repeat regions in another iteration of the exSPAdes module to construct the strain consensus sequence.

5.2.6 RNA-seq Assemblers

5.2.6.1 Velvet-Oasis Assembler

The Oases assembler [119] takes the output of the Velvet assembler as input. The first three stages of Velvet are used to produce an assembly. Oases processes the assembly produced by Velvet and the read mappings onto the contigs. The first step of the Oases assembler is to correct the contigs produced by Velvet. It uses an error correction module similar to Velvet’s to detect and merge bubbles in the assembly graph that have similar paths. For nodes that have multiple outgoing edges, Oases removes an edge if it has less than 10% of the read coverage of all outgoing edges from that node.

Once contig correction is completed, Oases constructs scaffolds from the contigs. Paired-end reads and spanning reads that overlap with multiple contigs are used to create connections between contigs. These connections are weighted according to the number of spanning and paired-end reads supporting the connection. The scaffolds are filtered and connections with less than three spanning reads and/or paired-end reads are removed. Connections that only have paired-end reads as support are also subjected to an additional filter that compares the number of supporting pair-end read pairs to the expected number of supporting paired-end read pairs given the contigs’ coverage and estimated distance apart. Paired-end read only connections that have less than 10% of expected paired-end read support are removed.

Once the scaffolding filtering is complete, Oases clusters the contigs into connected components that the authors denote as loci. Each loci is transitively reduced. Following transitive reduction, the Oases assembler analyzes the loci graph structure to produce the isoform sequences
for each gene.

### 5.2.6.1 Trinity Assembler

The Trinity assembler [120] was developed specifically for RNA-seq assembly and consists of three modules: *Inchworm, Chrysalis*, and *Butterfly*. The first module, Inchworm, rapidly assembles the reads into contig transcripts using a greedy k-mer approach. Inchworm only retains one best representative per contig transcript. To assemble the RNA-seq data into transcript contigs the Inchworm module constructs a k-mer dictionary. This dictionary is used to identify and remove erroneous k-mers. In order of frequency, k-mers are then selected from the dictionary to seed contig assemblies. These seeds are extended in both directions with k-1 overlapping k-mers. If multiple k-1 k-mer options are available, Inchworm selects the k-mer with the highest coverage. The output of the Inchworm module is one representative isoform per gene. Alternate isoforms are retained as partial sequences.

The Chrysalis module clusters the contigs output by the Inchworm module. A de Bruijn graph is constructed for each connected component cluster. Individual connected components correspond to individual genes and their isoforms. The Chrysalis module then assigns the reads to the connected component with which they share the most k-mer words.

Finally, the Butterfly module reconstructs transcript isoforms from the de Bruijn graph connected components. The Butterfly module accomplishes this by merging linear stretches in the connected components while trimming low coverage edges that are likely the result of sequencing error.

### 5.2.6.2 SOAPdenovo-Trans

The SOAPdenovo-Trans [121] assembler is an extension of the SOAPdenovo assembler that integrates concepts from Trinity and Oases. Like SOAPdenovo, SOAPdenovo-Trans utilizes the de Bruijn graph as its primary assembly platform. Once the de Bruijn graph is constructed, it is trimmed for errors. Instead of removing all k-mers that have a coverage value below a given
threshold, SOAPdenovo utilizes the coverage of adjoining neighbors in the graph to determine coverage cutoffs. If an edge has less than 5% of the coverage of its adjacent neighbors, then it is trimmed from the graph. By utilizing local coverage to trim erroneous edges, the SOAPdenovo-Trans assembler avoids trimming low coverage gene transcripts from the de Bruijn graph.

Once the graph is trimmed the contigs are generated using previous SOAPdenovo methods. Single-end and paired-end reads are then mapped back onto the constructed contigs. For genomic read data sets, high read can be used to determine repetitive contigs. However, this is not applicable in RNA-seq as some transcripts might be highly expressed. To address this issue, the SOAPdenovo-Trans assembler simply removes all contigs with length less than 100 bp. Similar to Oases, the SOAPdenovo-Trans assembler analyzes graph topology to produce the assembled transcripts per gene. Gaps in the assembly resulting from removing contigs less than 100 bps are filled using paired-end reads.

5.2.6.3 TopHat and Cufflinks

The TopHat [122] software tool is a RNA-seq read mapping application. It utilizes the Bowtie [123] read mapping tool to map RNA-seq reads onto a reference genome. Reads that do not map to the reference may cover a splice junction. The TopHat tool breaks unmapped reads into segments that are remapped to the reference genome. If groups of segments belonging to the same read map with a gap between 100 bp to several hundred kilobases, TopHat classifies this read as a splice junction spanning read. The locations of the splice site are estimated from the mapping of the read.

The Cufflinks software tool assembles the reads mapped by TopHat and Bowtie into transcript isoforms. As there are often multiple ways to construct isoforms from the RNA-seq read mappings, the Cufflinks tool attempts to construct the most parsimonious assembly. It minimizes the number of transcript fragments reported given a cluster of RNA-seq reads with multiple possible isoform assemblies. The Cufflinks tool also quantifies the abundance level of
each isoform that it assembles. Additional tools are also provided with the TopHat/Cufflinks package. The Cuffmerge tool merges two or more Cufflinks assemblies together. The Cuffdiff tool takes two or more Cufflinks assemblies and reports the differentially expressed genes between them.

5.2.7 Long Read Assemblers

5.2.7.1 CANU Assembler

The Canu assembler [124] is the successor to the CABOG assembler and was developed specifically for the 3rd generation long reads produced by Nanopore or PacBio. The Canu assembler has three stages, correction, trimming, and assembly.

The Canu assembler employs MinHash k-mer search to determine potential overlaps between reads. K-mers are probabilistically selected from a read according to their frequency in the read data set and in the read being overlapped. The MinHash algorithm creates a compressed k-mer fingerprint of a read for efficient read overlapping. These compressed sketches are compared to determine overlaps between reads.

Similar to the CABOG assembler, the CANU assembler constructs a best-overlap graph from the read overlaps. Multiple rounds of read error correction are conducted before overlaps are loaded into the graph because of the high error rates of long read technology (10-20%). Contigs are constructed from the best overlap graph in an approach similar to CABOG.

5.2.7.2 Hinge Assembler

The Hinge Assembler [125] is an overlap graph based tool for the assembly of long reads produced by PacBio or Nanopore technologies. Overlap relationships between reads are determined by the DALIGNER algorithm [14]. The HINGE assembler does not perform any type of error correcting before read overlapping.

Once alignments are obtained, the HINGE assembler utilizes the alignments to detect and remove chimeric reads from the read data set. The HINGE assembler also utilizes the overlap
relationships to identify any reads that might be part of a repeat region. Potential starting and ending points of repeats are annotated on reads identified as part of a repeat. If a read is determined to cover the entire repeat, its annotation is flagged as a bridged repeat.

After the reads are annotated with repeats, the HINGE assembler builds a graph where nodes are beginning and ending repeat annotations. Nodes that correspond to annotations that are from the same repeat are connected by an edge. The authors of HINGE construct a contagion algorithm on the annotation graph. First, small connected components are removed from the graph as they might be erroneous. Second, if an annotation is marked as a bridged, then all of its neighbors in the graph become “infected” and are also marked as bridging annotations. All annotations that have marked as bridging are “killed” by the algorithm and not processed further. The remainder of the connected components in the graph belongs to unbridged repeats. The contagion algorithm iterates again and kills annotations that belong to reads that have a neighboring read that overlaps the repeat a greater length.

Once completed, the remaining reads become hinges in the assembly graph. There is a single in-hinge and out-hinge per repeat sequence. These hinges aid the assembler to correctly traverse the graph in its greedy traversal algorithm. Global graph topology is also utilized to resolve repeats if necessary.
CHAPTER 6
PROJECT GOALS AND OBJECTIVES

Bioinformatics has emerged as a powerful technological field that has revolutionized the biomedical sciences through the unique fusion of the traditional academic disciplines with an atmosphere of cutting-edge scientific research. Bioinformatics applies computational and mathematical modeling to solve exciting yet challenging problems in the analysis, storage, and transformation of biological data. As demonstrated in chapters two and three, one of the most challenging problems in bioinformatics is the analysis and assembly of next generation sequencing read sets. Next generation sequencing technologies yield several hundred megabases to even up to a terabase of sequencing data in a single run.

While the assembly of the reads produced by NGS remains a difficult task, it is the process of extracting useful knowledge from these relatively short sequences that is quickly becoming one of the most exciting and challenging problems in Bioinformatics. Most current assemblers rely on the assembly graph as the foundational model for representing NGS reads. However, the assembly graph is primarily used to organize NGS data for assembly purposes, even though as a structural model it could be used as the basis of an expanded model to capture genomic structural features intrinsic to the input dataset.

6.1 Problem Statement

Several assembly tools were reviewed in chapter five. The end goal of most assemblers including those that were reviewed is to produce longer contigs with the major focus being on assembly only. Most assemblers are one-size-fits-all and do not take input data characteristics into consideration. Next generation sequence assembly is an aggregative process during which information is lost as reads are merged into contigs. As reads are flattened into contigs, there is a loss of local and global connectivity information. Locally, a decision is made on how reads will
be joined and merged together. Once reads are merged any alternative alignment is lost. Globally information is lost regarding how groups of reads are connected to one another; these global connections might reveal biologically relevant information such as the orientation of a repeat or another genomic feature. Here is a list of several problems that this dissertation addresses.

- **Problem 1. Pure assembly versus data analysis and feature extraction**

  The graph is a very powerful mathematical model that has an extensive mathematical foundation and history. The majority of assemblers utilize the assembly graph (which is defined here as any type of graph used for assembly, ie - the de Bruijn, the overlap graph, etc.) as a scaffold, lending structure and organization to the read data set during the assembly process. Assembly tools follow typical steps of graph construction, graph error trimming, and graph traversal to construct contigs in an assembly focused process. This pure assembly approach is very basic in nature and does not take advantage of the wealth of biologically relevant information that can be captured by the assembly graph. As discussed in the introduction, assembling read data sets into contigs is an information lossy process. During assembly the assembler is forced to make a choice when assembling ambiguous regions; either terminating the current contig being assembled or choosing one of multiple assembly possibilities. This choice and subsequent loss of information can lead to misassembly and incorrect results. In contrast, the assembly graph is capable of maintaining all information that is available regarding a read data set [85]. It may be beneficial to avoid assembly altogether and directly extract features from the assembly graph.

  Previous research has hinted at the capability of a graph model to capture genomic structure in a read sequencing data set. For example, in [126] graph-based clustering was used to characterize repetitive sequences in plant genome next generation sequencing data sets. It also briefly noted by the authors of the MetaVelvet [116] that ambiguous nodes in the
assembler graph (ie- those nodes that are part of two different paths in the assembly graph) might correspond to features such as transposable elements or conserved sequences such as rRNA sequences. However, this concept of ambiguous nodes corresponding to genomic features is not explored any further. There is very limited study on how the structure of the assembly graph is related to the genomic structure hidden in a read data set. The majority of assemblers do not take advantage of the rich topography of the assembly graph, primarily utilizing it as a data structure to organize a read data set being assembled. The assembly graph could be a very powerful support for a wide range of genomic feature extraction and data-mining applications.

• **Problem 2. Data independent assembly of next generation sequencing reads**

  The majority of assembler tools are black boxes that are independent of their input data characteristics. Most assembly tools are computationally focused with the data itself having little influence during the assembly process. All assemblers will produce output given an input data set. Blindly applying a generic assembly tool might produce incorrect results. It is unreasonable to expect that an assembler that is a one-size-fits-all approach will perform successfully for every sequencing data set that it is applied to, especially if the sequencing data sets have widely varying characteristics. The Assemblathon 2 [94], which is a competitive benchmarking study evaluating the performance of assemblers for different data sets, noted that assemblers were not consistent in their performance. Some assemblers performed much better for some data sets in comparison to others. This highlights the importance of understanding an assembler’s performance as a function of the input data set characteristics. The ability to select an assembler with prior knowledge regarding its performance for a next generation sequencing data set with specific characteristics would be beneficial.
Assembly tools also rely on a large set of parameters to control the assembly process. Limited study has been conducted for parameter optimization as a function of input data characteristics. The large selection of parameters available to researchers with little guidance on data-specific parameter values can make it difficult to optimize parameter values for different data types. The most critical parameters for assembly are the minimum overlap length for overlap graph assemblers and the k-mer length for de Bruijn graph assemblers. Assembler parameters, including the minimum overlap and k-mer length, are tuned over the entire read data set. It is reasonable to assume that a single parameter value might not be appropriate for the entire read data set. For example, an area of a genome might have lower read coverage compared to another region of genome that had higher read coverage. A large minimum overlap or large k-mer value might be appropriate in high coverage regions; however, in low coverage regions this parameter choice may lead to gaps in the final assembly. Some assembly tools do try to address this issue by iterating over a range of minimum overlap or k-mer values[108][109]; however, this iterative process is still applied uniformly to the entire read data set. It is a critical need for the development of tools that are flexible and adaptable to input data set characteristics for an intelligent, customized, domain-specific approach for assembly.

- **Problem 3. Massive data set sizes and naïve parallel assembly approaches**
  High performance computing has emerged to play a key role in addressing the computational burden of several steps of the read assembly process, including parallel read alignment and distributed graph modeling of read overlap relationships. As sequencing costs continue to drop, high performance computing will become increasingly more essential for the computationally efficient and timely analysis of read data available to researchers. Multiple short read assembly methods have successfully applied parallel computing approaches to meet large dataset processing requirements. However, most parallel
applications developed for assembly are naïve in nature. Naïve approaches for developing parallel NGS applications may achieve a high degree of parallelism while unnecessarily consuming computational resources and high levels of energy, a critical concern as the rate of NGS data production continues to increase exponentially.

When applying parallel computing techniques, most assembly tools focus primarily on speeding up the assembly process to provide faster results. Parallel computing has been applied to both the read-overlapping step of assembly as well as to the graph construction and manipulation step of assembly. For example, the Canu assembler [124] can parallelize the computation of read overlaps into multiple jobs. The PCAP assembler [127] also applies parallel computing to determine overlaps between reads. Other assembly tools partition the assembly graph across multiple processors so that graph trimming and other graph algorithms can be conducted in parallel. The ABySS assembler distributes the de Bruijn graph across multiple processors to speed graph processing. The Ray assembler [107] also distributes the de Bruijn assembly graph across multiple processors.

The assemblers discussed in the previous paragraph apply parallel computing techniques in a naïve approach with the major focus on speeding up the assembly process. Smart application of parallel computing can provide insights into the structure and characteristics of the data set on which it is applied. For example, if a read data set is split into multiple jobs for read overlapping and one of the jobs requires more time than other jobs for overlapping, then this subset of reads might contain a higher percentage of repetitive sequence. Further, it might be beneficial to utilize data set specific characteristics of the input data set to improve parallel computing algorithms.

High performance computing will allow assembly tools to be scalable to the needs of researchers in the biomedical community in a smart, energy efficient manner that utilizes and provides insights into biological properties of NGS data.
6.2 Research Purpose

In this research, a new innovative graph approach for assembly called Focus is proposed that not only assembles NGS reads but also is capable of mining valuable biological knowledge in the process. Unlike previous methods that utilize a single graph model for representing a read data set, the Focus assembler utilizes a multiset of graphs that can represent a read data set across a spectrum of granularity. In this dissertation, four aims are presented. First, a discussion of the development and implementation of the Focus platform is presented. Second, in aim 2, the multilayer graph model is adapted as a data-mining support in addition to its role as a platform for assembly. It is demonstrated that a wealth of biologically relevant information can be discovered from the multilayer graph model’s structural features including ambiguous graph nodes, which have previously been considered stumbling blocks for many NGS tools. In addition, we explore graph characteristics that lead to the discovery of biologically relevant features in NGS datasets. We also investigate how the assembly graph under the proposed approach can be used to analyze comparative genomics data. The ability to directly extract information from the NGS reads and structural features of their assembly graphs will provide a powerful method of analyzing genomic data and lead to new biological discoveries. Aim 3 demonstrates how the assembly process can be customized to produce better assembly results. This is achieved by shifting the classical interval overlap graph model to a tolerance graph model and customizing individualized parameters for each read based on its read neighborhood characteristics rather than utilizing a global parameter setting. Finally in aim 4, intelligent and domain-specific high performance computing methods implemented in the Focus assembler are presented. First a graph-partitioning scheme for the multilayer graph model that incorporates domain-specific information into the partitioning process is presented. This is followed by the introduction of an energy-aware parallel approach for determining read overlaps.
6.2.1 **Algorithm Overview**

The Focus algorithm consists of five steps. First the Focus preprocessor generates reverse complements of the input read dataset and splits the reads into subsets for parallel read alignment. The preprocessor also provides options for fixed-length and quality based read trimming. In the read alignment step of algorithm, Focus performs an all-against-all comparison of the read subsets generated by the preprocessor to search for potential alignments. Any overlap alignments found in the read alignment stage are used to create the initial overlap graph. The next step of the algorithm is the construction of the graph spectrum. In this step, Focus uses heavy edge matching and node merging to create a spectrum of graphs $G_0, G_1 \ldots G_n$ representing increasingly coarser levels of information granularity. In the fourth step, Focus backtracks through the graph spectrum starting with the simplest graph $G_n$ to select nodes that have been determined to be the best representatives of their corresponding read clusters. These representative nodes are used to construct a hybrid graph spectrum $G'_n, G'_{n-1} \ldots G'_0$ where $G'_0$ contains all of the representative nodes selected from $G_n, G_{n-1} \ldots G_0$. The hybrid graph $G'_0$ is processed with a graph-filtering algorithm to remove transitive edges and nodes whose corresponding read clusters assemble into contigs that are contained in or are identical to other contigs represented in the hybrid graph. The final trimmed hybrid graph $G''_0$ provides a very concise and highly accurate representation of the structure of the read dataset [128].

6.2.2 **Specific Aims**

- **Aim 1. Develop an innovative graph-theoretic algorithmic platform**

  Next generation sequencing technologies are capable of producing massive amounts of information in the form of millions to billions of short sequencing reads. Due to this reason, the assembly graph can become extremely large and complex as read dataset size grows, presenting a challenge to the assembly and data-mining process. Previous assembler tools utilize a single graph to model a read data set. A single graph provides a single, static view of a read data set. In contrast, we have developed a novel multilevel graph model that maintains
a consistent representation of the underlying sequence structure by the application of graph-
coarsening and partial assembly tactics. This set of graphs modeling across a spectrum of
information granularity is very flexible, allowing for the assembler to ‘Focus’ on different
local to global views of the read data set. Once the multilayer graph set is constructed, the
Focus assembler integrates the multilayer graph layers to construct the hybrid graph. The
assembler maintains local read overlap information in the original overlap graph. The
multilayer graph series of increasingly coarsened graphs reveal global structure in an input
read data set. The hybrid graph represents a concise yet information rich representation of the
input read data. This dissertation shows that the hybrid graph is a strong support for data
mining applications.

This information rich structural model is a component of a complete assembly pipeline
that we have developed which also includes a read quality preprocessor and parallel read
aligner. The read preprocessor is capable of trimming and filtering a given input read data set.
The parallel read aligner divides the input read data set into jobs that are sent to multiple
processors for independent read processing. Following the read overlapping, the assembler
constructs the multilayer graph model. Once the construction of the multilayer graph set is
complete, the assembler integrates the layers to construct the hybrid graph. The hybrid graph
can then be used for data mining and read analysis and/or can be error trimmed and used as
an assembly platform. This multilayer and information rich model will be a strong analysis
platform, allowing for the efficient extraction of graph structural elements relevant to
biologically important features.

• **Aim 2. Apply graph data-mining to extract biologically relevant features**

  The assembly graph is a structural model that captures the relationships between reads in
a read data set. It is reasonable to expect that structural features in the assembly graph might
represent structural phenomenon in the genomic sequence being assembled. The assembly
A novel multilayer graph model has been developed that allows for the flexible analysis of next generation sequencing data sets. The hybrid graph is the integration of the multilayer graph set that captures features of the input read data set. This dissertation proposes that multi-path nodes, which are usually seen as obstacles to the assembly process, as well as other structurally important features in the assembly graph represent a wealth of biologically significant information. We have developed graph-mining techniques that are applied to the hybrid graph model to reveal key structural trends corresponding with biologically significant features, including rRNA sequences and transposases. This novel graph-mining approach is also applied in a comparative study of the gut microbiome of Crohn’s disease patients and healthy individuals to extract differences in antibiotic resistance genes in the two sample groups. The distribution of antibiotics resistance genes was dramatically different between the two sets of microbiome samples. This study is extended to extract additional transposase-associated features from the hybrid graph, again demonstrating differences between microbiomes from Crohn’s disease patients and those from healthy individuals. The ability to directly extract information from the structural features assembly graph will provide a powerful method of analyzing next generation sequencing data in addition to mapping and assembly approaches.

Aim 3. Customize the assembly process to produce improved assembly results

To address current assembly challenges, a wide variety of short read assemblers have been developed. However, dataset size, underlying sequence complexity, and sequencing error create much ambiguity in the assembly process. In addition, it can be difficult to
determine which assembler might be appropriate for a particular assembly domain. Many assemblers apply a one-size-fits-all approach to widely varying data set types. This is unlikely to produce optimal results for all data types. For the second aim of this dissertation, customization of the assembly process is explored. First a case study is provided, evaluating the performance of assemblers on bacteriophage data. Results show that several assemblers misassemble circular bacteriophage data and data obtained from bacteriophages that replicate via rolling circle replication. This highlights the need for domain-specific assemblers that take the characteristics of the input data set into account.

Next, this dissertation focuses on the customization of assembly parameters according to individual read characteristics. To achieve this goal, the assembly graph is formulated as a new theoretical graph model called the tolerance graph. In this graph, each read is mapped to a node in the graph and overlap relationships are mapped to edges. The difference between this tolerance graph model and the overlap graph model is that each individual read is assigned its own customized minimum overlap parameter. This individualized minimum overlap parameter is dependent on the characteristics of the read that it is assigned too. Characteristics of the read’s neighborhood, such as read coverage, are used to tune the minimum overlap threshold. Parameter optimization conducted using a tolerance graph approach for the assembly process results in improved assembly results. The ability to customize the assembly process will result in an assembler that is capable of adapting to wider variation of short read data input such as read coverage, GC content, and error rate variation.

• **Aim 4. Develop intelligent high performance computing approaches for NGS assembly**

Due to the extremely large size of NGS data sets, several steps of the Focus algorithm have been implemented to run in high performance computing environments. A naïve parallel approach that utilizes a large number of processors may expend large amounts of
computational energy while minimally improving runtime. However, unlike previous approaches that are naïve in nature, the Focus algorithm is being developed to exploit energy aware computing for computational resource optimization while meeting researcher needs as well as integration of data characteristics to improve graph distribution across computational resources. In a collaborative project, we applied an energy aware scheduling model to run the read alignment portion of the Focus assembler. Results demonstrated the energy-aware scheduling model was able to adjust the number of computational processors to meet read overlapping deadlines, while minimizing energy expenditure.

Multilevel graph partitioning utilizing graph coarsening and the Kernighan Lin algorithm has been implemented to partition the hybrid graph across processors. Previous assemblers distribute the assembly graph across multiple computational processors in a data independent manner. In a data dependent approach, the Focus algorithm injects known features of NGS data to improve the partitioning process. Biological knowledge is utilized to identify which clusters of nodes in the original overlap graph should belong to the same partition and do not need to be refined with the Kernighan Lin algorithm, greatly improving runtime. Further, this data injection is beneficial for downstream analysis; it is shown that the data-dependent partitioning of the assembly graph can reveal insights into the community structure of microbiome data sets. Finally, several parallel graph trimming and traversal algorithms were implemented and successfully applied on the distributed hybrid graph.

In conclusion, we have developed an NGS assembly and analysis tool called Focus that allows for the extraction of biological knowledge through graph mining and is highly customizable according to researcher needs. High performance computing will allow the Focus algorithm to be scalable to the needs of researchers in the biomedical community in a smart, energy efficient manner that utilizes biological properties of NGS data.
CHAPTER 7

FOCUS: A NEW MULTILAYER GRAPH MODEL FOR SHORT READ ANALYSIS

7.1 Introduction

This chapter focuses on the organization and structure of the Focus assembler. In this research, we have developed a graph theoretic modeling approach that is able to capture multiple snapshots of local and global read relationships across a spectrum of granularity for the end goal of graph-based read analysis rather than pure assembly[129][128]. Unlike previous methods that rely on a single graph model, this algorithm, which is called Focus, constructs a multiset of graphs, $G_i = (V_i, E_i)$ that as a whole, can model the reads from localized relationships between individual reads to global relationships between subsets of reads within the dataset. Given an original overlap graph $G_0 = (V_0, E_0)$, this algorithm uses heavy edge matching (HEM) [130] and node merging to progressively generate a sequence of reduced graphs $G_0, G_1 \ldots G_n$ where $|V_0| \geq |V_1| \geq \ldots \geq |V_n|$. Each node in a reduced graph $G_i$ represents a cluster of one or more reads and maps to one or two child nodes that have been merged from $G_{i-1}$.

We describe a new, hybrid graph set $G'_0, G'_1 \ldots G'_n$ from $G_0, G_1 \ldots G_n$ using a novel backtracking method. Starting with the coarsest graph in the multilayer graph set $G_n$, we select nodes by assembling the read clusters represented by each node. If a read cluster assembles into a single contig, then its corresponding node is selected as an appropriate representative of that cluster and is mapped to a new node in $G'_n$ which is not decomposed into its child nodes in $G'_{n-1}$, $G'_{n-2} \ldots G'_0$. A node whose read cluster does not assemble into a single contig is also mapped to a new node in $G'_n$ but is not selected to be a representative of its read cluster. After evaluation is complete on $G_n$ the algorithm backtracks to the finer grained graph $G_{n-1}$ to determine if the child nodes of the nodes not selected from $G_n$ are better representatives of their corresponding read clusters. If a node in $G_{n-1}$ is determined by assembly evaluation to be an appropriate

---

1 Material in this chapter has been previously published: J. Warnke-Sommer and H. Ali, “Graph Mining for Next Generation Sequencing: Leveraging the Assembly Graph for Biological Insights “, BMC Genomics. http://creativecommons.org/licenses/by/4.0/
representative of its corresponding read cluster, it will not be decomposed into its child nodes in \( G_{n-2}, G_{n-3}, \ldots, G_0 \). This process is repeated until nodes have been selected from each graph level in \( G_0, G_1, \ldots, G_n \). The final graph constructed, \( G'_0 \), is formed from subsets of nodes from different graph information levels which have been determined to be the best representatives of the read clusters to which they correspond. Thus the graph \( G'_0 \) maximizes graph conciseness while minimizing error rates of node read clusters making it an ideal model for the large but highly complex metagenomics datasets.

Different regions in a set of genomes may not be optimally represented at the same graph information level. Thus a heterogeneous set of nodes is used to accurately capture genomic structure. We propose that this hybrid graph model can be used to extract various graph features that correspond to biologically significant information embedded in the input data such as features associated with shared regions across related genomes in metagenomics samples. The hybrid graph itself may be viewed as a dynamic rather than static structure since the multilayer graph set maintains all read overlaps and global cluster relationships from the initial overlap graph and coarser graph levels. Any of the nodes in the hybrid graph can easily be decomposed into finer grained representations as needed by using the information stored in the multilayer graph set. Thus, Focus presents a highly flexible and robust model that can be used for an analysis and data-mining approach in addition to being used as a robust platform for assembly.

To test Focus, we apply the model to simulated metagenomics datasets in four sets of experiments. First a multilevel graph set is constructed to represent each of the metagenomics datasets and determine the read clustering error rates. Next, a hybrid graph set is created from each of the multilayer graph sets and the clustering error rate and graph size of \( G'_0 \) of the hybrid graph set is compared to \( G_n \) of the original multilevel graph set. We then compare the performance of the model against the single genome assembler MIRA [131] and the metagenomics assembler MAP [132].
7.2 Methods

The Focus algorithm consists of five steps including read preprocessing, pairwise read alignment, multilayer graph set generation, multilayer graph set integration and generation of the hybrid graph, and hybrid graph trimming. Here we provide a brief overview.

- **Read preprocessor:** The Focus preprocessor generates reverse complements of the input read data set and splits the reads into subsets for parallel read alignment. The preprocessor also provides options for fixed-length and quality based read trimming.

- **Pairwise read alignment:** In the read alignment step of algorithm, Focus performs pairwise comparison of the read subsets generated by the preprocessor to search for potential alignments. Any overlap alignments found in the pairwise read alignment stage are used to create the initial overlap graph.

- **Multilayer graph set:** The next step of the algorithm is the construction of the multilevel graph set. In this step, Focus uses heavy edge matching and node merging to create a set of graphs $G_0, G_1 \ldots G_n$ representing increasingly coarser levels of information granularity.

- **Hybrid graph:** In the fourth step, Focus backtracks through the multilevel graph set starting with the most reduced graph $G_n$ to select nodes that have been determined to be the best representatives of their corresponding read clusters by local assembly analysis. These representative nodes are used to construct a hybrid graph set $G'_n, G'_{n-1} \ldots G'_0$ where $G'_0$ contains all of the representative nodes selected and integrated from the multilevel set $G_n, G_{n-1} \ldots G_0$. We call $G'_0$ the hybrid graph.

- **Hybrid graph trimming:** The hybrid graph $G'_0$ is processed with a graph-filtering algorithm to remove transitive edges and nodes whose corresponding read clusters
assemble into contigs that are contained in or are identical to other contigs represented in the hybrid graph. The final trimmed hybrid graph $G_0$ provides a concise but highly accurate and feature rich representation of the structure of the read data set [128].

7.2.1 Preprocessor and Input Format

Focus accepts both fasta and fastq formatted reads. Focus requires the user to specify the number of subsets to divide the read file into for parallel read alignment. Once Focus receives the input reads and specified number of subsets, it generates the reverse complements of the input reads. The preprocessor also includes both fix-length and quality based read trimming. While generating the reverse complements, the preprocessor will first trim the 5' and 3' ends of each read and the corresponding 3' and 5' ends of its generated reverse complement with fixed lengths $l_1$ and $l_2$ respectively that have been provided by the user. We provide this option so that the user can remove any known adapters or tags, which may or may not be the same length, present on the 5' or 3' end of the reads. After fixed length trimming is completed on a read, the preprocessor will then apply quality based trimming to its 3' end and to the corresponding 5' end of its generated reverse complement. Given a user-provided window length of $w$ and minimum average quality value $q$, the preprocessor will slide the window from the 3' end to the 5' end of the read until the average quality value of the window is greater than $q$. The reads in the subsets are then concatenated and indexed by a succinct dictionary structure [133]. In this structure, each nucleotide and corresponding quality value are compressed into a single byte. The read subsets are now ready for processing by the parallel read aligner. The read subsets are now ready for processing by the parallel read aligner. The parallel read aligner will be discussed in more detail in later chapters.
Figure 7.1: Focus assembly and analysis pipeline. (A) Read preprocessor. The 5’ and 3’ read ends are trimmed using quality values and/or by a fixed length specified by the user. Reverse complements of reads are generated and the processed read dataset is split into subsets for processing by the parallel read aligner. (B) Parallel read alignment. Reads are split into subsets and aligned using a suffix-array seed based search and extend method. (C) Multilayer graph set. Iterative heavy edge matching and node merging is used to create a spectrum of graphs. (D) Hybrid graph set. Best representative nodes are selected at each graph level using partial assembly to create the hybrid graph $G_0$. (E). Hybrid graph trimming. Transitive edges and redundant nodes are trimmed from $G_0$. 2016 BMC Genomics reprinted with permission
7.2.2 Multilayer Graph Set

The initial overlap graph constructed from the read overlaps produced during the parallel read alignment process may be extremely large if there are several hundreds of thousands to several millions of reads represented in the overlap graph. This would make the data mining process very difficult, as the resulting graph would be very complex. Heavy edge matching and node merging is applied to reduce the overlap graph, creating multiple graph levels representing different levels of granularity. This section describes the multilevel graph set construction in greater detail. An illustration of this process can also be found in Figure 7.2. First, parallel merge sort orders the initial edges produced by the alignment algorithm by query read id; edges with the same query read id are ordered by descending overlap length. Any duplicate edges with the same edge points as other edges in this sorted edge list are removed during the merge-sort process. This edge list is loaded by a graph data structure. For more information regarding the time-space complexity of the multilevel graph set construction and implementation of the foundational data structures used by Focus please see chapter eight. This initial overlap graph is denoted as $G_0$. In this graph, each node represents a single read. An edge between two nodes in this graph represents an overlap relationship between their corresponding reads. Each edge in $G_0$ maintains the overlap length and identity score of its corresponding read overlap relationship. The weight of the edges in $G_0$ is defined to be the overlap length. This graph is the least reduced and most granular graph in the graph set $G_0, G_1 \ldots G_n$ and is the foundation on which the other graphs in the set are built. Each node in $G_1, G_2 \ldots G_n$ represents a cluster of nodes in $G_0$. Two values are recorded for each node in the multilevel graph set to reflect the characteristics of its corresponding cluster in $G_0$. The cluster node weight of a given node in $G_1, G_2 \ldots G_n$ is the number of nodes belonging to its corresponding cluster in $G_0$. The cluster edge weight of a node in $G_1, G_2 \ldots G_n$ is the sum of the weights of the edges induced by the nodes in its corresponding cluster in $G_0$. The nodes in $G_0$ are assigned a cluster node weight of one and a cluster edge weight of zero since each node in $G_0$ corresponds to an individual read.
Heavy edge matching and node merging are used to create the multilevel graph set. The heavy edge matching heuristic [130] forms a maximal matching with preference for edges with larger edge weight by matching each node $v_i$ to an adjacent unmatched node neighbor $v_j$, such that the edge $(v_i, v_j)$ has the largest edge weight in the set of edges incident to $v_i$ that are not already part of the matching. Focus employs a modified heavy edge matching scheme to reduce the overlap graph. During the heavy edge matching process, the graph is iterated over in a user-defined number of passes such that all nodes in the graph are visited and nodes with larger maximum edge weights are visited in earlier passes. Let $v_i$ be a node that the algorithm is currently visiting. The algorithm will iterate through the edges of $v_i$ in the order of decreasing edge weight $v_i$ to find a potential match. Let $v_j$ be a node adjacent to $v_i$. If $v_j$ has not been matched to any previous node, the algorithm will examine the edge $(v_i, v_j)$ to see if it meets user-defined thresholds, discussed next, for inclusion in the heavy edge matching. First it examines the weight of the edge, which in $G_0$ is defined as the overlap length. If the weight of $(v_i, v_j)$ does not meet user requirements for minimum edge weight, then the search through the edges of $v_i$ is terminated and $v_i$ is left unmatched. If the weight of $(v_i, v_j)$ is greater than the user defined threshold for minimum edge weight, then $v_j$ passes the first test. The second threshold is the density of the super node $v_z$ that would result from the merging of $v_i$ and $v_j$. The density of $v_z$ is defined as follows.

$$
density(v_i, v_j) = \frac{\text{density}(v_z) =} {2^* (\text{ew}[v_i] + \text{ew}[v_j] + w(v_i, v_j))}$$

$$\frac{(\text{nw}[v_i] + \text{nw}[v_j])^* (\text{nw}[v_i] + \text{nw}[v_j] - 1)^{1/2}}{\text{nw}[v_i] + \text{nw}[v_j] + w(v_i, v_j)}$$

where $\text{ew}$ is the cluster edge weight, $\text{nw}$ is the cluster node weight, and $w$ is the weight of the edge
$(v_i, v_j)$. Here the density is the summed weights of the intra-cluster edges of the cluster in $G_0$ represented by $v_z$ divided by the total number of potential edges in that cluster if it was complete. This parameter controls the compactness of the merged cluster and ensures that many of the reads represented by that cluster overlap with one another. If the density of the super node that would be produced by merging $v_i$ and $v_j$ is greater than the user-provided threshold, then $v_i$ is matched to $v_j$. If the minimum threshold is not met, then the search through the edges of $v_i$ for a node neighbor that meets the minimum overlap and density thresholds continues. If none are found, then $v_i$ remains unmatched.

After the matching process is completed on $G_0$, nodes that are a part of the matching are merged to their selected partners to form super nodes in the graph $G_1$. Nodes that were unmatched in $G_0$ are also mapped to new nodes in $G_1$. Edges that were selected during the matching process are removed in $G_1$ since their endpoints are merged into a single super node. Any parallel edges in $G_1$ are combined into a single edge and their edge weights are added together. As follows, each edge in the multilevel graph set will represent the summed weight of the inter-cluster edges of the clusters in $G_0$ represented by the endpoints of that edge. Heavy edge matching and node merging is applied on $G_1$ to produce $G_2$. This process continues until the ratio of nodes matched to graph size falls beneath a user threshold, producing a multilevel set of graphs $G_0, G_1, \ldots, G_n$. The graph $G_n$ is used to relabel the nodes in $G_0$ to form a new overlap graph $G_{\text{final}}$: any nodes co-occurring in a cluster represented by a super node in $G_1, G_2 \ldots G_n$ will be consecutively labeled in $G_{\text{final}}$. This allows the nodes in $G_{\text{final}}$ belonging to a cluster represented by a super node in $G_1, G_2 \ldots G_n$ to be loaded into memory concurrently by the algorithm for processing.
Figure 7.2: A) Multilayer graph set. $G_0$ is the most granular assembly graph created from all of the read overlap relationships generated during read alignment. Each read is assigned to a node in $G_0$ and overlap relationships are assigned to edges. Weights on edges reflect the length of the overlap relationship. Heavy edge matching and node merging is applied to create a spectrum of graphs. Clusters of reads are formed as nodes are merged at each graph level.

B) Hybrid Graph Set. Starting with the simplest graph, in this case $G_3$, Focus attempts to assemble the read clusters represented by each node. If the reads assemble into a single contig, then their corresponding node is selected as that cluster’s best representative. All nodes that are selected by Focus in $G_3$ as well as nodes not selected are used to create $G'_3$ of the hybrid graph set. If a node is not selected in $G_3$, then its children nodes in the next graph level, in this case $G_2$, will be evaluated. The graph $G'_2$ will be created from the nodes selected from $G_3$ and $G_2$ as well as from the nodes that were not selected in $G_2$. We denote the final graph $G'_0$ as the hybrid graph as it will contain all of the best representatives from $G_n \ldots G_0$. In the above right figure, graph level integration of $G_1 \ldots G_1$ better represents a split in the assembly graph. 2016 BMC Genomics reprinted with permission.
7.2.3 Hybrid Graph Set

After the graph coarsening process is completed, the algorithm will have produced a graph set $G_0, G_1, \ldots, G_n$ representing the original read data set at different levels of information. However, not all sequence regions will be best represented at all graph levels. A node from a reduced graph found later in the multilevel graph set might be sufficient for representing a simple unique genomic region. In contrast, more complex genomic structures might be better represented by the more detailed graphs earlier in the graph set. For example in Figure 7.2A, a branch point in original overlap graph is over reduced in graph $G_3$ in the multilevel graph set. However, this branch point is captured at more granular graph levels. To address this issue, best representative super nodes are selected and integrated from multiple graph levels to create a new hybrid graph that is a highly concise yet accurate representation of the input data set. This section describes how a hybrid graph set $G'_0, G'_1, \ldots, G'_n$ is constructed from the multilevel graph set $G_0, G_1, \ldots, G_n$. The algorithm creates the hybrid graph set by selecting best representative super nodes from the original multilevel graph set beginning with $G_n$ and iterating to $G_0$. A best representative super node is defined as a node selected from the most reduced graph level as possible whose corresponding cluster of reads assemble into a single contiguous contig. If a read cluster does not assemble into a single contig, it might not be well represented by its current graph level. Backtracking to earlier graph levels may provide better node representatives of the reads in that cluster.

To select the best representatives, the algorithm first iterates through $G_n$. For each super node in $G_n$, its corresponding cluster subgraph in $G_{\text{final}}$ is loaded into memory. Focus employs graph-cleaning techniques first introduced by [103] and used commonly by many assembly tools. Short dead-end branches that are shorter than a user-provided threshold are removed from the
Figure 7.3: Hybrid graph trimming. For each node, if its corresponding contig can be mapped to a neighboring node’s contig then that node is removed from the hybrid graph. Transitive edges are also removed from the hybrid graph. 2016 BMC Genomics reprinted with permission.
subgraph. Small bubbles in the graph, which are two distinct paths in the graph that have the same beginning and ending nodes, are also removed by eliminating the least weighted path. The subgraph is then transitively reduced following the approach in [85]. If the resulting graph is a single path representing a contiguous contig, then the super node is selected as the best representative of that read cluster. The read cluster is assembled into a contig and recorded on file. After the iteration through the nodes of $G_n$ is complete, all selected best representatives are mapped to nodes in $G'_n$. Nodes that were not selected as best representatives are also mapped to nodes in $G'_n$. After the best representative selection on $G'_n$ is complete, the algorithm begins the super node iteration and assembly evaluation process on $G_{n+1}$. If a node in $G_{n+1}$ is a component node of a merged super node in $G_n$ that was previously selected by the assembly algorithm as a best representative, it will not be evaluated or included in $G'_{n+1}$ since its parent was already chosen as the best possible representative. The graph $G'_{n+1}$ is created from all of the best representatives selected from $G_n$ and $G_{n+1}$ as well as from the nodes that were not selected in $G_{n+1}$. Contigs assembled from the best representatives in $G_{n+1}$ are recorded to file. The graph $G'_{n+2}$ will be composed of best representative nodes selected from $G_n$, $G_{n+1}$, $G_{n+2}$ and the nodes that were not selected in $G_{n+2}$. This process is continued for $G_{n+3}$ … $G_0$. The final graph $G'_0$ will contain all best representatives selected from $G_n$, $G_{n+1}$ … $G_0$. We call this graph the hybrid graph since it is the integration of all graph information levels. As in the multilevel graph set, each edge in the hybrid graph set represents the summed total of the edge weights of the inter-cluster edges of the two clusters in $G_{final}$ corresponding to the endpoints of that edge. Please see Figure 7.2B for an example and chapter eight for more algorithmic details regarding the construction of the multilevel graph set and hybrid graph set.

7.2.4 Hybrid Graph Filter

Once the hybrid graph $G'_0$ is created, it is filtered to remove any redundant nodes whose corresponding contigs are contained within other contigs represented in the hybrid graph. For
each node in $G'_0$, the graph-filtering algorithm will load its corresponding contig into memory. If the length of the contig is less than a user provided threshold, then the filter will load each adjacent node’s contig into memory. The current contig is aligned against its neighboring contigs. If the current contig can be mapped to any of its neighboring contigs, then its corresponding node along with its incident edges will be removed from the hybrid graph as shown in Figure 7.3. Any transitive edges in the hybrid graph are also removed. After the filtering algorithm is complete, each node in the hybrid graph will represent either a homologous region shared between species, a sequence repeat, or a unique genomic region.

7.3 Results

We present the results of three experiments applied on twelve simulated metagenomics data sets. First, we generate a multilayer graph set for each of the metagenomics datasets and calculate the read clustering error rates. Second, we create a hybrid graph set from each of the multilayer graph sets and compare the clustering error rate and graph size of $G'_0$ of the hybrid graph set to $G_n$ of the original multilayer graph set. Finally, we conduct an assessment of the performance of Focus as an assembly tool against the MIRA single genome assembler and the metagenomics assembler MAP.

7.3.1 Simulated Data Sets

Twelve different metagenomics data sets were simulated with various characteristics. Half of the metagenomics datasets were generated from twenty closely related genomes selected from *Bacillus* and *Staphylococcus*. The remaining datasets were generated from twenty bacterial genomes selected at random. All reference sequences were downloaded from the NCBI RefSeq [134] database. Two distributions of abundance ratios were used for the datasets generated from closely related genomes and for the datasets generated from the randomly selected genomes. The first distribution used was the uniform distribution, where the abundances of each species in the dataset are equivalent. The second distribution used was the lognormal distribution, where the abundances of each species in the dataset are not equivalent. The lognormal distribution for
species abundances in metagenomics communities has been well supported by the literature [135][136]. For each distribution of the closely related genomes and randomly selected genomes, the program MetaSim [137] was used to simulate three 454 reads datasets with an expected read length of 400 base pairs and an error rate of one percent. The three datasets for each distribution were generated at different coverage levels with two hundred thousand, one million, or five million reads.

7.3.2 Computational Environment and Algorithm Configuration

Focus, MAP, and MIRA were run on the 40 TF Tusker computing cluster located at the Holland Computing Center [138]. The read alignment module of Focus was run on thirty compute nodes of the Tusker cluster with a minimum overlap length of fifty base pairs and minimum identity score of ninety percent. The graph coarsening and hybrid graph construction processes of Focus were run on a single compute node. The MIRA and MAP assemblers were each run on a single node of the Tusker computing cluster with default settings.

7.3.3 Multilayer Graph Set

Focus was applied to create a multilayer graph set for each of the twelve metagenomics datasets. To evaluate the ability of the multilayer graph sets to accurately represent the read datasets, we calculated the error rate of the read clusters generated at each graph level. For each graph level, read clusters are classified to a single species by majority read vote. A read is misclassified if its cluster is assigned to a species that the read did not originate from. The error rate of a graph level is calculated as the number of misclassified reads divided by the total number of reads in the dataset.

The error rates of the multilayer graph sets generated from the lognormal and uniform species distributions for the closely related genomes were similar. The error rates of the coarsest graphs constructed for the datasets containing one million or five million reads ranged from 5.3-5.9% (Fig. 7.4A). There was a decrease in the error rates for the read datasets containing two

---

2 Material in this chapter has been previously published: J. Warnke and H. Ali, “Focus: A New Multilayer Graph Model for Short Read Analysis and Extraction of Biologically Relevant Features“, The 5th ACM Conference on Bioinformatics, Computational Biology, and Health Informatics, © 2014 ACM
hundred thousand reads. The error rates of the coarsest graphs produced for these datasets ranged from 2.1-2.3% (Fig. 7.4A). Likely, smaller and less dense clusters are being generated by the multilayer graph set due to the low read coverage. The error rates for the maximally coarsened graphs for datasets generated from the randomly selected genomes was below .2% in all cases. This indicates that the algorithm is able to correctly cluster non-related sequences into distinct clusters. The graph-coarsening algorithm was able to substantially reduce the original overlap graphs representing the read datasets. Clearly, there is a greater benefit of graph coarsening and graph reduction with increasing coverage of the read dataset. The overlap graphs representing the five million read datasets were reduced an average of eighty fold. The overlap graphs for the one million read datasets were reduced an average of eleven fold. The overlap graphs for the two hundred thousand read datasets were reduced with an average of a two-fold reduction (Fig. 7.4B).

Fig. 7.4 demonstrates the relationship between the iterative reduction of the overlap graph and the error rate. As graph simplification increases there is a loss of information as reads are merged into clusters. It would be beneficial to find the optimal graph level at which to represent a read dataset as simply as possible while minimizing clustering error rates. However, it is likely that not all regions of the genome will be optimally represented at the same graph levels. For example, a non-exact repeat may be better represented by finer grained graphs, which might allow for the extraction of repeat sub-structure that coarser graphs are not detailed enough to represent. In contrast, exact repeats or unique genomic regions may be sufficiently represented by a coarser graph. The next section demonstrates the ability of Focus to integrate multiple graph layers to produce a hybrid graph set.

7.3.4 Hybrid Graph Set

We present the results of creating hybrid graph set for the read datasets generated from lognormal and uniform distributions of closely related species. Fig.7.4 displays the error rates for
the nodes selected or rejected at each graph level. Fig. 7.5 displays the number of nodes selected from each graph level in the original multilayer graph sets. The majority of nodes are selected from the coarsest graph in each case. However, a second local maximum for node selection from graphs earlier in the multilayer graph set indicate that the algorithm is backtracking as needed. Results demonstrate that the clustering error rates for nodes selected are much lower than then those rejected by the algorithm, especially for the coarsest graph level where most of the nodes were being selected. For the one million read dataset with a lognormal distribution of closely related species, the error rate of the rejected nodes was lower than the error rate of the selected nodes at graph level \( G_7 \). However, Fig. 7.6 shows that very few nodes were actually selected from this graph level. These results demonstrate that the assembly evaluation algorithm is able to distinguish between nodes that are and are not good representatives of their corresponding read clusters.

For the one million read dataset with a lognormal distribution of closely related species, the error rate of the rejected nodes was lower than the error rate of the selected nodes at graph level \( G_7 \). However, Fig. 7.6 shows that very few nodes were actually selected from this graph level. These results demonstrate that the assembly evaluation algorithm is able to distinguish between nodes that are and are not good representatives of their corresponding read clusters.

For each of the hybrid graph sets, the graph \( G'_0 \) will contain all of the nodes selected from each of the graph levels in \( G_1, G_2 \ldots G_n \). The number of nodes and read clustering error rate of \( G'_0 \) for each of the read datasets is shown in table 7.1. Table 7.1 also displays the number of nodes and error rate of the coarsest graph \( G_n \) from the multilayer graph set. Results from table 7.1 demonstrate that the error rate of the graph \( G'_0 \) is substantially less than the error rate of the coarsest graph \( G_n \) while minimally increasing graph size. The read clustering error rates for the five million and one million read datasets range from 5.3-5.8% for the coarsest graph. For the hybrid graph, error rates for these datasets were reduced to 1.5 – 1.9%. 

Figure 7.4: Multilayer graph set construction results. (A) The multilayer graph set read classification error rates for the read datasets generated from lognormal and uniform distributions of closely related species. (B) The multilayer graph set node counts for the read datasets generated from lognormal and uniform distributions of closely related species. © 2014 ACM reprinted with permission.
Figure 7.5: Selected and rejected node read classification error rates. (A) Read classification error rates for the read datasets generated from the lognormal distributions of the closely related genomes. (B) Read classification error rates for the read datasets generated from the uniform distributions of the closely related genomes. The error rates of the nodes selected by the assembly evaluation algorithm are shown in black and the error rates of nodes not selected are shown in red. © 2014 ACM reprinted with permission.
Figure 7.6 Selected node counts. The number of nodes selected from each graph level to construct the hybrid graph set are shown for the read datasets generated from the lognormal and uniform distributions of the closely related genomes. © 2014 ACM reprinted with permission
Table 7.1: Hybrid graph set and multilayer graph set comparison. © 2014 ACM reprinted with permission

<table>
<thead>
<tr>
<th>Dataset</th>
<th>$G_\theta$ Node count</th>
<th>$G_\psi$ Node count</th>
<th>$G_\theta$ Percent error</th>
<th>$G_\psi$ Percent error</th>
<th>Alignment Runtime (sec)</th>
<th>Multilayer Runtime (sec)</th>
<th>Hybrid Set Runtime (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>200,000 reads, lognormal</td>
<td>124,123</td>
<td>122,672</td>
<td>1.6%</td>
<td>2.1%</td>
<td>400</td>
<td>290</td>
<td>114</td>
</tr>
<tr>
<td>200,000 reads, uniform</td>
<td>113,324</td>
<td>110,914</td>
<td>1.7%</td>
<td>2.3%</td>
<td>487</td>
<td>315</td>
<td>139</td>
</tr>
<tr>
<td>1,000,000 reads, lognormal</td>
<td>145,773</td>
<td>99,920</td>
<td>1.9%</td>
<td>5.8%</td>
<td>4,349</td>
<td>2,522</td>
<td>1,539</td>
</tr>
<tr>
<td>1,000,000 reads, uniform</td>
<td>155,080</td>
<td>110,768</td>
<td>1.9%</td>
<td>5.6%</td>
<td>5,248</td>
<td>2,956</td>
<td>2,139</td>
</tr>
<tr>
<td>5,000,000 reads, lognormal</td>
<td>226,181</td>
<td>62,900</td>
<td>1.6%</td>
<td>5.3%</td>
<td>13,050</td>
<td>29,874</td>
<td>19,703</td>
</tr>
<tr>
<td>5,000,000 reads, uniform</td>
<td>208,363</td>
<td>62,499</td>
<td>1.5%</td>
<td>5.9%</td>
<td>28,388</td>
<td>34,367</td>
<td>23,475</td>
</tr>
</tbody>
</table>
At most, the total node count for the hybrid graph was 3.6 times larger than the node count of the coarsest graph in its corresponding multilayer graph set. The graph $G'_0$ retains most of the conciseness of the coarsest graph model $G_n$ while approaching the accuracy of the finer-grained graphs in the spectrum. Table 7.1 also displays the runtimes for the various components of Focus. The read alignment module, multilayer graph set construction module, and hybrid graph construction module all have reasonable runtimes with no component requiring more than ten hours of runtime for any of the datasets.

7.3.5 **Assessment of Focus as an Assembler**

The MIRA and MAP assemblers were applied to assemble the one million read datasets for the lognormal and uniform distributions of the closely related species. For the purpose of comparing the performance of these algorithms to the read clusters produced by the model, we treated the contigs produced by MAP and MIRA as read clusters. For each contig, we assigned its component reads to a single species by majority read vote. Reads that were assigned to a species that they did not originate from were counted as misclassified. For each dataset, the read clusters generated from the graph $G'_0$ produced by Focus were used in the assembly comparison. Results can be found in table 7.2.

The assembler MAP had the lowest error rate at .1% for both datasets. However, this was at the expense of neglecting to assemble nearly half the reads in each dataset. The MIRA assembler produced the largest contigs but also had the highest error rates. The results of these assemblers highlight the need to find a balanced assembly approach. Methods that are too stringent may produce results with a low error rate but also with low information content. Lenient approaches may produce more informative yet less reliable results. Our algorithm was able to find a balance between these two extremes. Focus produced the fewest number of singlets with 35,587 singlets for the lognormal dataset and 22,442 singlets for the uniform dataset. Focus also
Table 7.2: Assembly comparison. © 2014 ACM reprinted with permission

<table>
<thead>
<tr>
<th>Assembler</th>
<th>Dataset</th>
<th>Percent error</th>
<th>Number of contigs or clusters ≥ 1</th>
<th>Singlets</th>
<th>Runtime (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Focus</td>
<td>1 mil. lognormal</td>
<td>1.9%</td>
<td>123,331</td>
<td>22,442</td>
<td>8,410</td>
</tr>
<tr>
<td></td>
<td>1 mil. uniform</td>
<td>1.9%</td>
<td>120,493</td>
<td>34,587</td>
<td>10,343</td>
</tr>
<tr>
<td>MAP</td>
<td>1 mil. lognormal</td>
<td>.1%</td>
<td>176,502</td>
<td>428,621</td>
<td>286,080</td>
</tr>
<tr>
<td></td>
<td>1 mil. uniform</td>
<td>.1%</td>
<td>176,777</td>
<td>428,522</td>
<td>294,640</td>
</tr>
<tr>
<td>MIRA</td>
<td>1 mil. lognormal</td>
<td>2.6%</td>
<td>28,609</td>
<td>53,995</td>
<td>33,203</td>
</tr>
<tr>
<td></td>
<td>1 mil. uniform</td>
<td>2.0%</td>
<td>29,642</td>
<td>82,609</td>
<td>31,297</td>
</tr>
</tbody>
</table>
maintained a low error rate of 1.9% for both datasets. By the integration of the multilayer graph set, Focus was able to substantially simplify the graph modeling of the read dataset while still maintaining an overall low clustering error rate.

Due to the efficient nature of the algorithm and the adaptation of the read alignment module to be run in parallel, Focus greatly outperformed MAP and MIRA in regards to program runtime. The MAP assembler required over three days to assemble the one million read datasets, while MIRA required roughly three to four times the amount of time of Focus to analyze the read datasets.

7.4 Conclusion and Discussion

Focus is a new model that uses multilayer graphs to represent various aspects and relationships of short reads obtained from metagenomics applications. This model takes advantage of biological properties of the genomes sequenced to facilitate extraction of useful information along with the production of more accurate assembled genomes. It can be viewed as a modeling tool as well as a data-mining tool.

In this chapter, the Focus assembler was applied to twelve metagenomics datasets to create a multilayer graph set modeling the reads at various levels of information. Multiple layers of this spectrum were integrated to create a hybrid graph composed of a heterogeneous set of nodes which may represent single reads, medium, or large clusters of reads depending on node information level.

Results demonstrate that the multilayer graph set is able to represent the read dataset across multiple levels of information. As graph complexity decreases through the spectrum, there is a loss of information reflected by increasing read cluster error rates. The hybrid graph was able to preserve much of the simplicity of the coarsest graphs in the multilayer graph set while approaching the accuracy of earlier graphs in the spectrum. The hybrid graph’s ability to find a
favorable compromise between graph simplicity and information content makes it a powerful model for the analysis and extraction of biologically relevant features from metagenomics sets. Future chapters conduct further exploration of the hybrid graph as a data-mining tool applied across a wide variety of sample types.
CHAPTER 8

AN EFFICIENT OVERLAP GRAPH COARSENING APPROACH FOR MODELING SHORT READS\textsuperscript{3,4}

8.1 Introduction

In this chapter, a more technical presentation of the Focus assembly and analysis tool is presented. This chapter is mostly computationally based and can be read depending on the interest in the technical aspects of the Focus assembler. Supporting data structures and algorithmic details are described. First, the efficient graph data structure for holding read overlap information is reviewed. A technical description of the graph coarsening and graph traversal approach is then provided. It is described how graph coarsening and graph traversal is used to obtain a relabeling of the initial overlap graph that is used to efficiently order the nodes within the graph data structure. Following this, technical details regarding the hybrid graph multilayer set and hybrid graph construction are presented. Finally, the results of applying the Focus algorithm to cluster a simulated metagenomics dataset are discussed. Results from evaluating the node relabeling and ordering scheme on three bacterial read datasets are also presented.

8.2 Methods

7.3.1 Graph Theoretic Model

The Focus assembler approach utilizes the overlap graph to model the reads and their overlap relationships. In the overlap graph approach for assembly, each read is mapped to a node in the overlap graph. If two reads overlap, then an edge connects their corresponding nodes in the graph. The alignment length and alignment identity information for each overlap is attached to the edges. An illustration of the overlap graph for read modeling in the Focus assembler can be found in Fig. 8.1.

\textsuperscript{3}Material in this chapter has been previously published: J. Warnke and H. Ali, “An Efficient Overlap Graph Coarsening Approach for Modeling Short Reads”, 2012 IEEE International Conference on Bioinformatics and Biomedicine Workshops (BIBMW) © 2012 IEEE
Figure 8.1: Graph theoretic model. (a) The overlap graph. (b) The interval graph. In the overlap graph, reads are mapped to nodes and read overlap relationships are mapped to edges. Overlap information such as overlap length and alignment identity can be assigned to edges as weights. Overlap lengths are assigned to the edges in (a). © 2012 IEEE reprinted with permission
In graph theory, there exists a class of graphs called the perfect graphs. The structural properties of perfect graphs and the performance of many graph theoretic algorithms on perfect graphs have been well characterized [139].

Recall that the overlap graph is similar to a perfect graph called the interval graph. A graph \( G \) is an interval graph if and only if each node in the graph can be assigned an interval on the number line such that if two intervals overlap then their corresponding nodes are connected by an edge in \( G \). The connection between overlap and interval graphs can be made if reads are envisioned as intervals on the number line. Indeed, if the read overlap relationships are perfect without missing or false-positive data, then the resulting graph would be an interval graph. In the case of a circular genome, the resulting graph would be a circular-arc graph, a generalization of the interval graph.

Notice that there is a natural left-to-right (right-to-left) ordering on the interval set in Fig. 8.1.b that can be extended to the nodes in graph \( G \). By definition, all interval graphs are chordal graphs and therefore do not have any chordless cycles greater than three [139]. Let \( L \) be an integer labeling of the nodes of an interval graph formed by considering the endpoints of its corresponding intervals from left to right. The chordal properties of the interval graph insure that distant sections of the graph will not be connected by edges and that neighbors in the graph will have similar labels in \( L \). We take advantage of this property of the interval graph to efficiently order the nodes in a graph data structure following a labeling obtained via a graph coarsening and graph traversal approach.

8.2.1 The Graph Data Structure

The graph data structure implementation relies on a succinct dictionary structure to index both node and edge information [140]. Given an array of \( n \) bits, a succinct dictionary provides two functions to access the bit array. The first function, \( \text{rank} \), returns the rank of the \( i \)th position in the array, which is the summation of ones in the array up to that position. The second function,
select, returns the position of the $i$th bit set to one. The functions rank and select are, respectively, constant-time and nearly constant-time operations [140].

Before the succinct dictionaries are constructed, the overlaps produced by the overlap algorithm are ordered into an edge list by their source node (query read) labels first and then by their alignment lengths, from longest alignment lengths to shortest.

Two separate succinct data structures are used to index the node and edge information. The first succinct dictionary, node_index, contains a bit array of length $n$ where $n$ is the number of nodes in the overlap graph. If a node in the overlap graph has incident edges, then its corresponding bit is set to one. If a node has no incident edges, then its corresponding bit remains set to zero.

The second succinct dictionary, edge_index, contains a bit array of length $m+1$ where $m$ is the number of edges in the overlap graph. For each node that contains edges in the overlap graph, a bit in the array is set to one to index the start position of that node’s edges in the edge list. The last bit in the array is set to one.

Two additional arrays of 32-bit integers are used to store edge information. The first array, dest_nodes, stores each edge’s destination node’s label in the order of the edge list. The source node labels are available through the two succinct dictionary data structures and do not need to be stored explicitly. The second array, edge_info, stores overlap length and overlap identity information in a single 32-bit integer per edge in the order of the edge list.

The two succinct dictionaries along with the two accompanying integer arrays are used to efficiently retrieve information from the graph. In particular, we will examine the efficiency of the following graph functions:
• **Edge_bounds(node u).** This function returns the starting position of the edge set of a node u in the edge list in nearly constant time. Let \( p = \text{node_index}.\text{rank}(u) \). The starting position of the edge set of u in the edge list will be \( \text{edge_index}.\text{select}(p) \). The upper bound of the edge set will be \( \text{edge_index}.\text{select}(p+1) \).

• **Edge_source(edge e).** This function returns the label of the source node of an edge e with edge list position \( p \) in nearly constant time. Let \( q = \text{edge_index}.\text{rank}(p+1)-1 \). The label of the source node of edge e will be \( \text{node_index}.\text{select}(q) \).

• **Edge_dest(edge e).** This function returns the label of the destination node of an edge e with edge list position \( p \) in constant time. The label of the destination node of edge e will be \( \text{dest_nodes}[p] \).

Fig. 8.2 provides an overview showing how the succinct dictionaries, \( \text{node_index} \) and \( \text{edge_index} \), and 32-bit integer arrays, \( \text{edge_dest_nodes} \) and \( \text{edge_info} \), are organized.

The succinct dictionary \( \text{node_index} \) requires \( O(n) \) bits of storage space, where \( n \) is the number of nodes in the overlap graph. The succinct dictionary \( \text{edge_index} \) requires \( O(m) \) bits of storage space, where \( m \) is the number of edges in the overlap graph. The \text{rank} and \text{select} functions require 25% and 37.5% additional space, respectively, for each succinct dictionary data structure [140]. The integer arrays \( \text{dest_nodes} \) and \( \text{edge_info} \) each require \( 32m \) bits of storage space, where \( m \) is the number of edges in the overlap graph.

### 8.2.2 Graph Coarsening

Graph coarsening is used to reduce the size and complexity of the overlap graph and to form clusters of similar reads [133]. The hierarchical nature of the coarsening scheme allows for information to be captured at various granularities of graph complexity.
Figure 8.2: Graph data structures. The graph data structure consists of two succinct dictionaries, `node_index` and `edge_index`, and two 32-bit integer arrays, `dest_nodes` and `edge_info`. In `node_index`, `i` is the index of the source nodes, `B_i` is the value of the bit array at `i`, and `Rank_i` is the value of `Rank(i)`. In `edge_index`, `j` is the edge index, `B_j` is the value of the bit array at `j`, and `Rank_j` is the value of `Rank(j)`. In `dest_nodes`, `D_j` is the index of the destination node of the edge with index `j` and in `edge_info`, `W_j` is the weight of the edge with index `j`. The process flow for the `edge_bounds` function, which returns the boundaries of a node’s edge set in `edge_index`, is shown above. © 2012 IEEE reprinted with permission.
In the first step of the graph coarsening stage, the edges produced by the overlap algorithm are ordered into an edge list by their source node labels first and then by their alignment lengths, from longest alignment lengths to shortest. A parallel merge-sort algorithm is used to order the edges. Any duplicate edges are removed from the edge list.

Given a graph $G_n = (V_n, E_n)$, Heavy Edge Matching (HEM) [130] is used to find a maximal matching that has a large edge weight in $G_n$. The Focus assembler computes the HEM by visiting each node in sequence and examining its edges in the order of largest to smallest edge weight. If a node $u$ has an unmatched neighbor $v$ and the matched pair meets certain requirements that are discussed later, then $u$ is matched to $v$ and the search through its edge set is terminated. For the purpose of giving the nodes with the heaviest edges matching priority, the matching process is conducted iteratively. A user-defined number of passes over the graph is used to match the nodes with the heaviest edges first followed by nodes with smaller edge weights. The complexity of HEM is $O(|E|)$; however, since the edges of each node are ordered from largest to smallest edge weight, it is unlikely that all edges of the graph will be visited during matching. An example of graph coarsening using matching can be found in Fig. 8.3.

Four integer arrays are used to store important information about the matching obtained in $G_n$. The first integer array we will consider is the `node_map`. The nodes that constitute a matched pair in $G_n$ are merged into a single node in $G_{n+1}$. Let $(u, v)$ be an edge in a matching $M$ on graph $G_n$, then $\text{node\_map}[u] = \text{node\_map}[v] = z$, where $z$ is the label of the combined node in $G_{n+1}$. Nodes that are not a part of $M$ are also mapped to a new node label in $G_{n+1}$. The second integer array is `node_map_inverse`, which is simply the inverse of `node_map`. This array is used to recover node clusters in subsequent steps of the algorithm. Let $z$ be a node in $G_{n+1}$, then $\text{node\_map\_inverse}[2\times z] = u$ and $\text{node\_map\_inverse}[2\times z+1] = v$, where $(u, v)$ is a matched node pair in $G_n$. If $z$ consists of a single node $u$ from $G_n$, then $\text{node\_map\_inverse}[2\times z] = u$ and $\text{node\_map\_inverse}[2\times z+1] = -1$. 
Figure 8.3 Graph coarsening. A maximal matching of size three has been constructed on a weighted graph $G_0$. Matched nodes in $G_0$ are merged to form a new graph $G_1$. Each edge with end points that are matched nodes in $G_0$ is removed in $G_1$. Edges b. and c. are merged into a single edge in $G_1$ and their edge weights are summed. The matching and node merging process is repeated on graph $G_1$ to form $G_2$. © 2012 IEEE reprinted with permission
The remaining two integer arrays are node_weights and edge_weights. For each node in a graph G_n, the array node_weights stores the number of child nodes in G_0 descended from that node. For each node in G_n, the array edge_weights stores the combined edge weights of the edges induced by its child nodes in G_0.

During the matching process on a graph G_n, the arrays node_weights and edge_weights are used to calculate the edge density for each merged node in G_{n+1}. Let w(u, v) be the weight of the edge connecting a matched node pair in a matching M on G_n. The edge density of the node z in G_{n+1} resulting from the merging of u and v in G_n is calculated with the following formula.

\[
\text{edge\_density}(u, v) = \text{edge\_density}(z) = \frac{2 \times (\text{ew}[u] + \text{ew}[v] + w(u, v))}{(\text{nw}[u] + \text{nw}[v]) + ((\text{nw}[u] + \text{nw}[v]) - 1)},
\]

where \(\text{ew}[u] = \text{edge\_weights}[u]\) and \(\text{nw}[u] = \text{node\_weights}[u]\).

If during the matching process a node u has an unmatched neighbor v and \(\text{edge\_density}(u, v)\) is less than a user-provided threshold, then u will not be matched with v and the search through the edge set of u for an appropriate neighbor will continue. The second parameter that is used to control the matching process is the minimum edge weight threshold. If during the search through the edge set of a node u, a neighbor v is found such that \(w(u, v)\) is less than the minimum edge weight threshold, then the search is terminated and u is left unmatched.

After the matching process is complete, the edges of G_{n+1} are created from the edges of G_n. Each edge \(e = (u, v)\) in G_n, is relabeled to create a new edge \(e_{\text{new}} = (\text{node\_map}[u], \text{node\_map}[v])\) in G_{n+1}. If \(\text{node\_map}[u] = \text{node\_map}[v]\), then \(e_{\text{new}}\) is discarded. Parallel merge sort is used to order the edges of G_{n+1} into an edge list, by the edges’ source node labels first and then by their edge weights, from largest edge weights to smallest. Any duplicate edges are combined into a single edge and their weights are summed together.
After the edge list is created, the matching process is begun on the new graph. This cycle of matching and node merging is continued until the ratio of the number of nodes matched to graph size falls beneath a user-provided threshold. The final result of the graph coarsening scheme is a series of graphs \( G_0, G_1, G_2 \ldots G_n \), where \( |V_0| > |V_1| > |V_2| > \ldots > |V_n| \).

### 8.2.3 Graph Relabeling

This section describes how graph coarsening and maximal path extraction is used to find a relabeling of the original overlap graph \( G_0 \) that places neighboring nodes close to one another in the graph data structure. Let \( G_n = (V_n, E_n) \) be the final graph produced by the graph coarsening algorithm and \textit{labeling} be the integer array used to hold the new node labeling scheme. The nodes of \( G_n \) are visited randomly and used to seed maximal paths in \( G_n \). Let \( u \) be a node chosen to seed a maximal path \( P = \{u_1, u_2\ldots u_n\} \) in \( G_n \). The node \( u \) is marked as visited and the edge set of \( u \) is iterated through from largest to smallest edge weight. The first unvisited node \( v \) found in the edge set of \( u \) is marked as visited and added to \( P \). A new search begins through the edge set of \( v \) for the next available node to place in \( P \). This process is continued until \( P \) cannot be extended any further. When the extension of \( P \) is complete, the \textit{node_map_inverse} arrays are used to recover a set of clusters \( C = \{c_1, c_2, \ldots c_n\} \) of nodes from the original overlap graph \( G_0 \), where cluster \( c_i \) is a descendant of the \( i \)th node in \( P \). Each cluster \( c_i \) is visited following the order of \( P \) and its nodes are iterated through and relabeled. Let \( u_i \) be a node in \( c_i \), then \( \text{labeling}[u_i] = j \), where \( u_i \) was the \( j \)th node in \( G_0 \) to be relabeled. After the relabeling process is complete, a final graph \( G_{\text{final}} \) is created from the new labeling scheme and the edges of \( G_0 \). Each edge \( e = (u, v) \) in \( G_0 \), is relabeled to create a new edge \( e_{\text{new}} = (\text{labeling}[u], \text{labeling}[v]) \) in \( G_{\text{final}} \). Finally, parallel merge sort is used to order the edges into an edge list, by the edges’ source node labels first and then by their edge weights, from largest edge weights to smallest to form \( G_{\text{final}} \).
8.2.4 Read Cluster Generation

The arrays node_map_inverse; are used to recover the clusters of child nodes descended from nodes of a graph Gi, where i > 0. Let vz be a node in graph Gi. Observe that the child nodes vi and vj of vz in Gi+1 will be given by node_map_inverse;[2*vi] and node_map_inverse;[2*vj]. The child nodes of vi and vj can then be found by using node_map_inverse;+1. This process is continued until the child nodes of vz in G0 are recovered. Once the cluster of child nodes belonging to vz in G0 is generated, the cluster of reads assigned to those nodes from the read dataset can be recovered.

8.2.5 Hybrid Graph Set Construction

If the reads mapped to the child nodes of a node vz in a graph Gi, where i > 0, have been correctly clustered together, there is no need to decompose vz into its component child nodes in Gi-1. However, if the reads have not been clustered together correctly in Gi, it may be beneficial to split vz into its child nodes vi and vj in the more granular graph Gi-1.

The Focus algorithm identifies nodes beginning with the coarsest graph Gn and ending with G0 that are likely to contain correctly clustered reads. These selected nodes in Gi, G2 … Gn are used to create a set of hybrid graphs G’1, G’2 … G’n where each selected node in Gi is mapped to a new node in G’1 that is not decomposed into child nodes in G’1, G’2 … G’n. The algorithm accomplishes this by sequentially analyzing the subgraphs induced by the child nodes in Gi of each node in a graph Gi that does not have a previously selected ancestor in Gi+1 … Gn.

Each subgraph is processed with a subset of assembly algorithms including transitively reduction, trimming of dead-end paths, and bubble resolution. After the subgraph is processed, if the algorithm is successfully able to traverse the subgraph through a single path and if the number of dead end paths and bubbles removed from the subgraph are less than a user-provided maximum, then its corresponding node in G’ will not be split into its component nodes in G’1, G’2 … G’n (Fig. 2). An integer array hyb_map; is used to map nodes in a graph Gi to nodes in a

---

4 Material in this chapter has been previously published: J. Warnke and H. Ali, “Focus: A New Multilayer Graph Model for Short Read Analysis and Extraction of Biologically Relevant Features”, The 5th ACM Conference on Bioinformatics, Computational Biology, and Health Informatics, © 2014 ACM
hybrid graph $G'$. Before node selection from $G_1, G_2 \ldots G_n$ begins, the members of each $hyb\_map$ array are initialized to negative one. An integer value $nodes\_selected$ tracks the number of nodes selected by the algorithm.

Let $G_i$ be a graph currently being evaluated by the algorithm in the series $G_1, G_2 \ldots G_n$, where $|V_1| \geq |V_2| \geq \ldots \geq |V_n|$. The nodes of $G_i$ are visited in the order that their child nodes appear in the reorganized graph $G_{final}$. Let $v_z$ be the node that the algorithm is currently evaluating in $G_i$. If $hyb\_map_i[v_z] = -1$, then the parent of $v_z$ was not selected in $G_{i+1}$ or $G_i$ is the coarsest graph in the series and $v_z$ should be processed further. Let $k_z$ be the subgraph induced by the child nodes of $v_z$ in $G_{final}$. First, $k_z$ is transitively reduced using the method presented in [85]. Short dead-end paths less than a user-specified maximum are identified and removed from $k_z$. Simple bubbles less than a user-specified maximum are also identified and removed. Paths are created from the subgraph by randomly selecting a node that has not been explored and extending a path from the node in both directions until a branch point is reached or the path cannot be extended any further. All nodes in the path are then marked as explored. The traversal algorithm recursively selects unexplored nodes until all nodes in the trimmed subgraph are included in a path. If the traversal algorithm is able to successfully extract a single path from the $k_z$ and the number of nodes removed from $k_z$ by the trimming and bubble-popping procedures is below a user-provided maximum, then $v_z$ and its child nodes in $G_{i-1}, G_{i-2} \ldots G_0$ will be mapped to a single node $v_{hyb}$ in $G_1', G_{i-1}' \ldots G_0'$.

Recall that the child nodes of $v_z$ are found by $node\_map\_inverse[2*{v_z}]$ and $node\_map\_inverse[2*{v_z+1}]$. Let $v_i = node\_map\_inverse[2*{v_z}]$ and $v_j = node\_map\_inverse[2*{v_z+1}]$, then we set $hyb\_map_i[v_z] = hyb\_map_i[v_i] = hyb\_map_i[v_j] = v_{hyb}$, where $v_{hyb} = nodes\_selected$. The value of $nodes\_selected$ is incremented by one.

If $hyb\_map[v_z]$ is not equal to -1, then an ancestor of $v_z$ must have been selected in $G_{i+1} \ldots G_n$. Let $v_{hyb} = hyb\_map[v_z]$. Let $v_i$ and $v_j$ be the child nodes of $v_z$ in $G_{i-1}$ given by
node_map_inverse[2*v_z] and node_map_inverse[2*v_z+1]. We set hyb_map_i[v_i] = hyb_map_i[u_i] = v_{hyb}. The algorithm does not evaluate the node v_z any further.

After all the nodes of G_i have been evaluated, the array hyb_map_i is iterated through. Let the integer relabel = nodes_selected + 1. For p = 0 \ldots |V_i|, if hyb_map_i[p] = -1, we set hyb_map_i[p] = relabel and relabel is incremented by one.

Once the node selection and mapping process has completed, the edges of graph G'_i are generated from the edges of the graph G_i and the node mapping recorded by hyb_map_i. Each edge e_i = (v_i, v_j) of G_i is visited consecutively and mapped to a new edge e_{new} = (hyb_map[v_i], hyb_map[v_j]) in G'_i. After the generation of the edges of G'_i is complete, the edges are sorted by parallel merge-sort to form the edge list. As before, any edges with the same endpoints are merged and their weights are added together. All self-loops are discarded. The sorted edge list is indexed by graph data structures to form G'_i.

The final result is a hybrid graph set G'_i = G'_0, G'_1 \ldots G'_n. The graph G'_0 will contain each node that was selected by the assembly evaluation algorithm from G_1, G_2 \ldots G_n. Thus G'_0 will represent the read dataset as concisely as possible while minimizing read clustering error rates.

8.3 Results

In this section, the ability of the graph coarsening approach to cluster reads at various graph granularities is evaluated by applying the algorithm to a simulated metagenomics dataset. Second, the relabeling scheme’s ability to arrange the nodes such that many adjacent nodes are within close proximity to one another in the graph data structure is tested by applying the algorithm to three additional bacterial read datasets.

8.3.1 Metagenomics Clustering

Eight bacterial genomes were downloaded from the NCBI RefSeq database [134]. Genomes were selected at various levels of biological classification to highlight the algorithm’s
ability to reveal sequence similarities at different levels of granularity. Half of the bacterial genomes were selected from the phylum Firmicutes and the other half were selected from the phylum Actinobacteria. Pairs of the genomes were selected from the same order and all genomes were from different species.

The software package MetaSim [137] was used to generate 40,000 simulated 454 reads with an expected read length of 400 bp from each genome. The characteristics of each dataset can be found in table 8.1. The eight datasets were combined into a single dataset with a total of 320,000 reads.

Read overlapping of the simulated metagenomics dataset was conducted using eight nodes on a large commercial strength cluster called Firefly at the Holland Computing Center [138]. After read overlapping was completed, the graph coarsening algorithm was applied using the Firefly cluster to the set of read overlaps that was output by the overlap algorithm. The merge-sort component of the graph coarsening algorithm was ran serially on a single Firefly node. The minimum threshold for the ratio of the number of nodes matched to graph size was configured at .01. The graph coarsening algorithm was ran with a minimum edge density threshold of 50.

After each graph coarsening iteration, read clusters were recovered from the reduced graph. Each cluster was assigned a classification at the species, order, and phylum level as determined by majority read vote. The error rate of the cluster classification was defined to be the percentage of reads in the dataset that were assigned incorrectly to a cluster. To insure that clustering quality was not dependent on read ordering, ten random permutations of the metagenomics dataset were generated and the algorithm was applied to each. The mean error rates and standard deviations for the classifications at the species and order level for various graph coarsening iterations are shown in Fig. 8.4a. The error rates for classification at the phylum level are minimal and are not shown in Fig. 8.4a.
Figure 8.4: (a) Read classification. Read clustering and classification was performed on a simulated metagenomics dataset generated from eight bacterial genomes. The mean classification error rate and standard deviation is shown for each graph coarsening iteration. (b) Graph relabeling. Node relabeling was applied to three bacterial read datasets obtained from NCBI. The graph displays the percentage of edges whose node labels are within a specified distance one another. (c) Graph coarsening statistics. The number of nodes is shown for each graph coarsening iteration. © 2012 IEEE reprinted with permission
Table 8.1 Simulated datasets. © 2012 IEEE reprinted with permission

<table>
<thead>
<tr>
<th>Reference Accession #</th>
<th>Organism</th>
<th>Genome Length (bps)</th>
<th>Number of Reads</th>
<th>Average Read Length (bps)</th>
</tr>
</thead>
<tbody>
<tr>
<td>NC_012472</td>
<td>Bacillus cereus</td>
<td>5 269 628</td>
<td>40 000</td>
<td>445</td>
</tr>
<tr>
<td>NC_017138</td>
<td>Bacillus megaterium</td>
<td>4 983 975</td>
<td>40 000</td>
<td>440</td>
</tr>
<tr>
<td>NC_017999</td>
<td>Bifidobacterium bifidum</td>
<td>2 223 664</td>
<td>40 000</td>
<td>406</td>
</tr>
<tr>
<td>NC_014656</td>
<td>Bifidobacterium longum</td>
<td>2 265 943</td>
<td>40 000</td>
<td>408</td>
</tr>
<tr>
<td>NC_017465</td>
<td>Lactobacillus fermentum</td>
<td>2 100 449</td>
<td>40 000</td>
<td>461</td>
</tr>
<tr>
<td>NC_017486</td>
<td>Lactobacillus lactis</td>
<td>2 399 458</td>
<td>40 000</td>
<td>448</td>
</tr>
<tr>
<td>NC_011896</td>
<td>Mycobacterium leprae</td>
<td>3 268 071</td>
<td>40 000</td>
<td>413</td>
</tr>
<tr>
<td>NC_017523</td>
<td>Mycobacterium tuberculosis</td>
<td>4 398 812</td>
<td>40 000</td>
<td>420</td>
</tr>
</tbody>
</table>
Classification error rates remain below two percent for all graph iterations. Notice that as the coarseness of the graph increases, the percentage of misclassified reads increases as additional similar reads are clustered together. At earlier stages of the graph coarsening process, very few reads are misclassified at the order level. It is only until after the fourth graph coarsening iteration we begin to see similar reads at the order level being clustered together.

8.3.2 Node Relabeling

Three bacterial read datasets were downloaded from the NCBI sequence read archive [134]. The characteristics of these datasets can be found in table 8.2. Read overlapping was conducted using eight nodes on the Firefly computing cluster [138]. After read overlapping was complete, the graph coarsening algorithm was applied to each dataset. The minimum threshold for the ratio of the number of nodes matched to graph size was set at .01. The minimum edge density threshold was set to 50. After the graph coarsening and node relabeling processes were complete the node organization in the final relabeled graph was evaluated. Recall that the smaller the distance between two nodes’ labels, the closer they are to one another in the graph data structure. To investigate how close each node was to its neighbors in the graph data structure, we calculated the difference between the node labels of each edge. Fig. 8.4b shows the percentage of edges whose node labels are within a specified distance of one another. Note that even with the relatively straightforward approach of clustering and graph traversal to obtain a relabeling of the overlap graph, that nearly 96% of the edges from the Mycobacterium tuberculosis dataset had node labels within a distance of 1000 from each other. For the Escherichia coli dataset, nearly 70% of the edges had node labels that were within a distance of 1000 from each other. Almost 75% of the edges from the Staphylococcus aureus dataset had node labels that were within a distance of 1000 from each other.
Table 8.2: Bacterial Read Datasets. © 2012 IEEE reprinted with permission

<table>
<thead>
<tr>
<th>Accession #</th>
<th>Organism</th>
<th>Number of Reads</th>
<th>Average Read Length (bps)</th>
</tr>
</thead>
<tbody>
<tr>
<td>SRR060737</td>
<td><em>Escherichia coli</em></td>
<td>337 294</td>
<td>425</td>
</tr>
<tr>
<td>SRR060736</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SRR014924</td>
<td><em>Staphylococcus aureus</em></td>
<td>395 462</td>
<td>238</td>
</tr>
<tr>
<td>SRR039122</td>
<td><em>Mycobacterium tuberculosis</em></td>
<td>174 615</td>
<td>333</td>
</tr>
</tbody>
</table>

Table 8.3: Read Overlapping Runtime (8 nodes). © 2012 IEEE reprinted with permission

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Runtime (seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>Escherichia coli</em></td>
<td>3649</td>
</tr>
<tr>
<td><em>Mycobacterium tuberculosis</em></td>
<td>2026</td>
</tr>
<tr>
<td><em>Staphylococcus aureus</em></td>
<td>1013</td>
</tr>
<tr>
<td><em>Metagenomics (avg.)</em></td>
<td>3239</td>
</tr>
</tbody>
</table>

Table 8.3: Graph Coarsening Runtime (Serial merge-sort) © 2012 IEEE reprinted with permission

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Runtime (seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>Escherichia coli</em></td>
<td>397</td>
</tr>
<tr>
<td><em>Mycobacterium tuberculosis</em></td>
<td>110</td>
</tr>
<tr>
<td><em>Staphylococcus aureus</em></td>
<td>433</td>
</tr>
<tr>
<td><em>Metagenomics (avg.)</em></td>
<td>102</td>
</tr>
</tbody>
</table>
Fig. 8.4c displays the number of nodes at each graph iteration. The first graph coarsening iteration reduces the number of nodes in the overlap graph approximately by half. Large reductions in node numbers occur early in the graph coarsening process. After roughly seven iterations there are minimal reductions in graph size. Finally, table 8.3 displays the read overlapping run times and table 8.4 displays the graph coarsening and relabeling runtimes for the various datasets.

8.4 Conclusion and Discussion

In this chapter, the technical implementation of an efficient graph data structure and graph coarsening scheme for modeling next generation sequencing data has been presented. This approach is unique in that next generation sequencing datasets are represented at multiple levels of granularity with a series of graph models in contrast to a single overlap graph. This local-to-global view of the read dataset takes advantage of the structure and organization provided by graph models and of the ability of clustering to reveal information in large-scale complex data. Metagenomics results demonstrate this local-to-global picture of the read dataset as increasing numbers of similar reads from high biological classifications were clustered together as graph coarsening progressed. In future chapters it is shown that extracting features from the hybrid graph, such as nodes with large node weights or edge densities, nodes involved in cycles, or nodes with skewed GC content, provides insights into the global features of the read dataset such as repetitive content and species composition. Small-scale details including individual overlap information are retained in the original overlap graph.

An efficient graph data structure for storing read overlap information obtained through graph coarsening, graph traversal, and node relabeling has also been introduced. In this relabeling scheme, we took advantage of the overlap graph’s similarity to the interval graph. Due to the chordal properties of the interval graph, most adjacent nodes in an overlap graph would likely share many of the same neighbors. The natural left-to-right ordering of the interval graph
suggested that traversing maximal paths through the reduced graph would find an ordering of the merged nodes that would be more likely to keep their child nodes in the original overlap graph close to one another in the graph relabeling. Results from applying the node relabeling scheme to three bacterial genomes demonstrated that 70 – 96% of the edges for each dataset had node labels that were a within a distance of 1000 from each other. This relabeling scheme not only efficiently organized the nodes in the graph data structure; it revealed node groups in the overlap graph that likely correspond to unique regions of sequence. This information can be used for further downstream analysis, including contig assembly. In contrast to single overlap graph based assemblers, the proposed Focus assembly algorithm utilizes a series of graphs to analyze both local and global read dataset characteristics.
CHAPTER 9

GRAPH MINING FOR NEXT GENERATION SEQUENCING: LEVERAGING THE ASSEMBLY GRAPH FOR BIOLOGICAL INSIGHTS

9.1 Introduction

In this chapter, the Focus assembler is evaluated in its capacity as a graph theoretic data-mining support. The hybrid graph is shown to be an excellent data-mining support that can be used to extract structural signatures associated with biological features and make novel biological discoveries. The Focus algorithm was applied for a study on read data sets obtained from the gut microbiomes of healthy individuals and individuals with Crohn’s disease. The overarching goal of this research was to explore the distribution of transposase genes, associated antibiotic resistance genes, and other transposase-associated genes across bacterial genera in the gut microbiomes of healthy individuals and individuals with Crohn’s disease. The approach and results in this manuscript might provide insights into candidate genera for which horizontal gene transfer of transposon sequences and associated antibiotic resistance genes has occurred. This study is divided into three specific aims.

1. *Demonstrate that repetitive and transposable elements are associated with node characteristics.* To facilitate efficient extraction of meaningful graph structures in this study, each node in the hybrid graph is assigned a Shannon’s index score to reflect the local diversity of the various species or sequence regions in which the sequence represented by the given node is conserved or repeated. The Shannon’s index captures the number of edges incident to a given node as well as their evenness or how equally their weights are distributed. In the hybrid graph, each node represents a contiguous sequence region. Edges represent overlap relationships that this sequence region has with other contiguous sequence regions. An edge between two given nodes is weighted
according to the summation of the read overlap lengths between reads composing the first sequence region with the reads composing the other sequence region. If a sequence region is repeated multiple times in a single genome or is present in multiple species, it might follow that its representative node in the assembly graph will have multiple in and out edges representing different sequence regions or species. In contrast, a node that is part of a single path in the assembly graph might be representative of a unique genomic region.

Bacterial transposons are mobile DNA segments that can independently replicate and insert themselves within the same chromosome or plasmid or into a different chromosome or plasmid [141]. They have been implicated in the horizontal transfer of genes between different bacterial species. Transposase and integrase sequences are often a part of transposable elements and are commonly involved in their transfer. The simplest of bacterial transposons is the insertion sequence (IS) element shown in Fig. 9.1A, which is composed of two inverted repeats flanking genes necessary for transposition. The rDNA operon is a prevalent large repeat class in microbial genomes, ranging from 1-15 copies per genome [142]. In this manuscript, it is shown that nodes assigned with transposase/integrase genes and rRNA operon DNA had a greater proportion of high Shannon’s index scores in comparison to nodes assigned with other gene categories from the SEED subsystems (q = 2.44x10^{-04}; paired Wilcoxon tests).

2. **Identify and characterize the phylogenetic distribution of antibiotic resistance gene classes associated with transposase/integrase sequences in healthy individuals and individuals with Crohn’s disease.** The human microbiome has been described as a reservoir for antibiotic resistance genes [143][144] and as a hot spot for horizontal gene transfer [145] between bacterial taxa. Antibiotic resistance genes are often found in
bacterial composite transposons, which are composed of two IS elements flanking a protein coding sequence region as shown in Fig. 9.1B [146], allowing their rapid spread between bacterial groups. Crohn’s disease is a chronic disorder where the gastrointestinal tract is inflamed [147]. Horizontal gene transfer has been suggested to be increased between pathogenic and commensal bacteria in inflamed gastrointestinal systems [148]. Furthermore, this population is more likely to be treated with antibiotic regimens for secondary complications such as bacterial overgrowth and abscesses [149]. Antibiotic use has been shown to increase antibiotic resistance in those to whom it is prescribed [150]. Due to these issues, it is important to understand the antibiotic resistance genes present in populations affected by Crohn’s disease.

In a novel graph-mining approach, the structure of the hybrid graph is used to identify transposase/integrases sequences that might be located in multiple sequence regions (i.e. repeated in the same genome or distributed across multiple species) according to their assigned Shannon’s index score. Local graph exploration of the neighborhood surrounding these transposase and integrase sequences reveal associated antibiotic resistance genes. Clustering transposase sequences based upon their phylogenetic distribution obtained from the hybrid graph revealed several differences between the Crohn’s disease and healthy data set. Most transposase genes in the healthy data sets were clustered into a large Bacteroides group significantly enriched for tetracycline, macrolide-lincosamide-streptogramin B, and beta-lactamase antibiotic resistance genes. Transposase genes in the Crohn’s disease data sets were more diverse across phylogenetic groups including an Enterococcus cluster significantly enriched for aminoglycoside, macrolide, and streptogramin antibiotics resistance genes. This approach reveals clusters of genera for which transposase associated antibiotic classes are enriched.

---

5Material in this chapter has been previously published: J. Warnke-Sommer and H. Ali, “Graph Mining for Next Generation Sequencing: Leveraging the Assembly Graph for Biological Insights “, BMC Genomics. https://creativecommons.org/licenses/by/4.0/
and may provide insight into candidate bacterial groups in which horizontal gene transfer has occurred.

3. **Identify and characterize all additional genes and functionalities that are associated with mobile elements and transposase genes in the Crohn’s disease and healthy data sets.** This aim extends the analysis conducted for the previous two aims to examine all gene features associated with transposase sequences in the Crohn’s disease and healthy metagenomics data sets. Antibiotic resistance genes are commonly known to be in association with transposase sequences. However, it would be beneficial to examine additional genetic features associated with transposase sequences in Crohn’s disease and healthy gut microbiome samples. The ability to determine transposase associated gene differences in Crohn’s and healthy gut microbiomes provides valuable insights into potential niche adaptations of bacterial species in gut environments afflicted with Crohn’s disease.

Results demonstrate the knowledge that can be obtained from structural features of the assembly graph. Nodes annotated with several genetic features that are distributed across multiple species or are often present in multiple copies (rRNA) have a significantly greater proportion of high Shannon’s index scores than other nodes in the hybrid graph. This reflects a greater number of unique sequences that overlap with the genomic regions of these particular nodes. Graph mining is also useful for comparative studies, allowing for the identification of distinct differences in composition of transposase associated antibiotic resistance genes in the Crohn’s disease and healthy data sets. Additional transposase associated gene features are significantly different in Crohn’s disease versus healthy gut microbiomes. Crohn’s disease samples had significantly greater hits to several functional categories, including beta-glucoside metabolism,
Figure 9.1: A) The insertion sequence (IS) is the smallest transposon present in bacterial chromosomes and plasmids. It is composed of two inverted repeats flanking genes necessary for transposition. B) The composite transposon is composed of two IS elements flanking a central protein coding DNA region. This central region often contains genes for antibiotics resistance. 2016 BMC Genomics reprinted with permission.
maltose and maltodextrin utilization, and heme, hemin uptake and utilization systems in gram positives. The ability of the hybrid graph to reveal multiple genera that a given transposase sequence is present within may provide insights into the flow of horizontal gene transfer and antibiotic resistance gene spread in metagenomics samples. Graph mining is a powerful method of next generation sequencing data analysis in addition to assembly and read mapping methods.

9.2 Methods

This chapter focuses primarily on the data-mining aspects of the Focus assembler. For a description of the Focus assembly algorithm please see chapters seven and eight.

9.2.1 Hybrid Graph Data-Mining

The hybrid graph is used for mining and extraction of biologically significant features, since it provides the most concise, yet accurate structural view of the read data set that could be obtained from integrating the multilevel graph set. In this graph, the degree of a given node can provide much information about the characteristics of the sequence region from which its corresponding reads were derived. If a node has a single pair of in and out edges, it is possible that this node is from a uniquely represented genomic region. In contrast, if a node has several in and out edges, this might indicate that the node represents a sequence region that is repeated throughout a genome or is shared between multiple species. The number of edges incident to a node might reflect the number of diverse sequences that its corresponding genomic region is present within.

The first aim of this research is to show that repetitive and mobile elements are associated with node characteristics. Shannon’s index is very popular for measuring biological diversity [151], however; it has not yet been applied for characterizing sequence diversity captured by graph structures in assembly graphs. Shannon’s index encompasses both the edge richness and edge weight evenness of a given node. Edge richness refers to the number of edges incident to a
node. Edge evenness measures the distribution of weight across the edges. The formula for Shannon’s index is given by

\[ H = - \sum_{i=1}^{n} \frac{w_i}{W_{\text{total}}} \ln \left( \frac{w_i}{W_{\text{total}}} \right), \]

where \( n \) is the number of incident edges, \( w_i \) is the weight of the \( i \)th edge, and \( W_{\text{total}} \) is the total weight of all incident edges. As seen in Figure 9.2, a greater number of edges and an equal distribution of edge weights increases a node’s Shannon index score. The maximum Shannon’s index score that can be assigned to a node \( v \) is \( \ln(n) \), where \( n \) is the number of edges incident to \( v \).

The score of a node that has two edges with similar large weights and multiple edges with very small weights will not be very different from the score of a node with only two edges with similar weights. Thus any possible spurious edges with small edge weights relative to the edge weights of the other incident edges will not greatly impact a node’s Shannon index score. In the results section, it is demonstrated that repetitive and mobile elements are associated with graph structure that is captured by the Shannon’s index. Fig. 9.3A provides an example illustrating homologous regions shared between two genomes and corresponding graph structure.

The second aim of this research is to extract transposase genes that are present in multiple sequence regions and to identify which genera they are distributed in. Antibiotic resistance genes associated with these transposase sequences are also mined from the hybrid graph. In this section, it is discussed how, for each given transposase gene present in multiple sequence regions, the hybrid graph can be used to identify which genera the transposase gene is distributed in. This section also discusses how the hybrid graph can be used to obtain transposase associated antibiotic resistance genes through local exploration in the hybrid graph. Observe that in Fig. 9.3B, the distribution of genera that a transposase sequence is shared across can be obtained by taxonomically classifying the sequences of the adjacent node neighbors of its corresponding node in the hybrid graph. Similarly, any antibiotics resistance genes that are associated with a given
**Figure 9.2: Shannon’s index scores.** Calculation of Shannon’s index scores. Notice that nodes with a greater degree have a higher Shannon’s index score. 2016 BMC Genomics reprinted with permission.
Figure 9.3: Genomic features and related graph structure. A) Genomes one and two share a region of sequence homology. In the hybrid graph this homologous region will be reduced to a single node (purple). Two paths representing the unique regions in genomes one and two preceding the 5’ end of the homologous region enter the reduced node. The two paths exiting the node represent the unique genomic regions in genomes one and two following the 3’ end of the homologous region. Blue represents genome one and red represents genome two. Observe that we can identify which species a given region is present in by analyzing its representative node’s neighbors in the hybrid graph. For example, obtaining species level classification for the nodes adjacent to purple node would identify in which species the region represented by the purple node was present. B) Genomes one and two share a homologous transposase sequence. Each of these genomes also contains a transposase associated antibiotics resistance gene. As in (A), the homologous region containing the transposase sequence is reduced to a single node shown here in purple. The nodes corresponding to the antibiotics resistance genes are colored black and blue represents genome one and red represents genome two. Graph exploration of the node neighborhood of the node representing the homologous transposase sequences will reveal antibiotics resistance genes associated with them. BMC Genomics reprinted with permission.
transposase sequence can be found by exploring the graph locally around its corresponding node. Fig. 9.3B provides an example of how local graph exploration can reveal antibiotics resistance genes associated with transposase sequences. The third aim extends this concept, mining all additional gene features associated with transposase sequences in the Crohn’s disease and healthy microbiome data sets.

9.3 Results

In this section the distribution of transposase genes and associated antibiotic resistance genes across bacterial genera in the gut microbiomes of healthy individuals and individuals with Crohn’s disease is characterized using graph mining techniques. To achieve this goal this study has been divided into three specific aims discussed previously in the background and methods.

1) Demonstrate that repetitive and transposable elements are associated with node characteristics

2) Identify and characterize the phylogenetic distribution of antibiotic resistance gene classes associated with transposase/integrase sequences in healthy individuals and individuals with Crohn’s disease.

3) Identify and characterize all genes and functionalities that are associated with mobile elements and transposase genes in the Crohn’s disease and healthy data sets.

The results are divided into four sections. First, a general overview of the data sets and an analysis of the distribution of genera present in the Crohn’s disease and healthy data sets are provided. This study was conducted to evaluate the characteristics of the data sets in the context of previous research. Statistically significant differences were found in the relative abundances of prevalent genera in the Crohn’s and healthy data sets.

Second, aim 1 is addressed. For aim 1, it is shown that a greater proportion of nodes annotated with repetitive and mobile elements are assigned high Shannon’s index scores
compared to nodes annotated with other gene categories. First an analysis and discussion regarding the distribution of Shannon’s index scores across the nodes of the hybrid graphs of the thirteen data sets is presented. This is followed by briefly exploring the characteristics of features associated with nodes with high Shannon’s index scores. The most common blastx hits to the NCBI blast database [152] for extremely high scoring nodes (the two highest scoring nodes for each data set) were to transposases and integrases (33.3% of all predicted genes). Gene and rRNA operon predictions, SEED subsystems [153], and ACLAME library [154] were then used to examine biological features associated with the remaining graph nodes. Nodes assigned with transposase/integrase genes and rRNA operon DNA had a greater proportion of high Shannon’s index scores in comparison to nodes assigned with other gene categories from the SEED subsystems (q = 2.44x10^-04; paired Wilcoxon tests).

Third, addressing aim 2, a comparative study of antibiotics resistance genes associated with transposase/integrase sequences present in multiple sequence regions in the Crohn’s and healthy data sets was conducted. In aim 1, it is demonstrated that a greater proportion of nodes annotated with mobile genetic elements and rDNA operons had high Shannon’s index scores compared to nodes annotated with other gene categories. For aim 2, transposase/integrase sequences found on nodes with high Shannon’s index scores are analyzed since they are likely to be present in multiple sequence regions. We identify all high degree nodes with Shannon’s index scores greater than one that had hits to transposases, identify which genera their corresponding contig sequences are present in, cluster the transposases according to their phylogenetic distribution, and determine if sequence regions associated with the transposases in the resulting clusters are enriched for antibiotic resistance genes. The transposase nodes in the Crohn’s data sets clustered into twenty sets and the nodes in the healthy data set clustered into ten sets. For each of these clustered sets, predicted genes in associated contigs were extracted and DIAMOND [155] was used to align the predicted genes to the CARD database of antibacterial resistance
genes [156]. Fisher’s exact test with FDR corrected p-values was applied to determine if any clusters were enriched with classes of antibiotics. Several of the transposase clusters generated in the Crohn’s disease and healthy control data sets were enriched with various classes of antibiotic resistance genes. This comparative study provides insight into the differences in the distribution and species composition of resistance genes in healthy individuals and Crohn’s patients, whose disease is associated with gut microbiome perturbation [157] and is often treated with antibiotic regimens for secondary complications such as bacterial overgrowth and abscesses [149].

Finally for aim 3, all genes and functionalities associated with transposase genes are characterized. Following the approach in aim 2, the transposases that fall on nodes with Shannon’s index greater than one are analyzed. The functionality of genes associated with these transposases is determined. Results demonstrate that some transposase-associated features are more prevalent in Crohn’s disease data sets than healthy data sets. The biological relevance several of the functionalities discovered to be associated with transposase sequences in the Crohn’s disease and healthy data sets is explored.

9.3.1 Data Sets

Thirteen data sets were downloaded from the NCBI sequence read archive [158]. Five of the data sets were sequenced from the gut microbiome of individuals with Crohn’s disease and eight of the data sets were sequenced from the gut microbiomes of healthy individuals. Table 9.1 shows the subject ids for each data sets and their phenotype information. Table 9.1 also displays the number of reads in each data set prior to read trimming as well as the number of processed reads produced by the Focus read preprocessor, which includes generated reverse complement reads. The Focus preprocessor was set to trim 20 bps off of the 5’ read ends and 50 bps off of the 3’ read ends to remove tags and adaptors. The minimum quality value for the quality based trimming was set to 25. Any read whose length fell below 75 bps was discarded from the processed data set. A hybrid graph was constructed for each individual data set.
Table 9.1: Data Set Characteristics. 2016 BMC Genomics reprinted with permission

<table>
<thead>
<tr>
<th>Subject ID</th>
<th>Phenotype</th>
<th>Sample</th>
<th>Runs</th>
<th>Total Reads</th>
<th>Processed Reads</th>
<th>Mapped (%)</th>
<th>Shannon’s Index</th>
</tr>
</thead>
<tbody>
<tr>
<td>33</td>
<td>Female, Crohn’s</td>
<td>SAMN008 29176</td>
<td>SRR49544, SRR497943, SRR497952</td>
<td>1775071</td>
<td>3478940</td>
<td>80.6%</td>
<td>1.91</td>
</tr>
<tr>
<td>58</td>
<td>Female, Crohn’s</td>
<td>SAMN008 29163</td>
<td>SRR497643, SRR497648, SRR497650</td>
<td>2049784</td>
<td>4025328</td>
<td>68.4%</td>
<td>1.64</td>
</tr>
<tr>
<td>92</td>
<td>Female, Crohn’s</td>
<td>SAMN008 29171</td>
<td>SRR497646, SRR497657, SRR504939</td>
<td>1950395</td>
<td>3848348</td>
<td>61.4%</td>
<td>1.73</td>
</tr>
<tr>
<td>104</td>
<td>Male, Crohn’s</td>
<td>SAMN008 2172</td>
<td>SRR497946, SRR497948, SRR497949</td>
<td>2175693</td>
<td>4284474</td>
<td>72.1%</td>
<td>1.97</td>
</tr>
<tr>
<td>68</td>
<td>Male, Crohn’s</td>
<td>SAMN008 29168</td>
<td>SRR497645, SRR497652, SRR497654</td>
<td>2084020</td>
<td>4113996</td>
<td>79.0%</td>
<td>1.49</td>
</tr>
<tr>
<td>763820215</td>
<td>Female, Healthy</td>
<td>SAMN000 78732</td>
<td>SRR063543, SRR063544, SRR063545</td>
<td>2395215</td>
<td>4744426</td>
<td>88.9%</td>
<td>0.59</td>
</tr>
<tr>
<td>764042746</td>
<td>Female, Healthy</td>
<td>SAMN003 6587</td>
<td>SRR063587, SRR063588, SRR063589</td>
<td>2260051</td>
<td>4463710</td>
<td>81.5%</td>
<td>0.56</td>
</tr>
<tr>
<td>809635352</td>
<td>Female, Healthy</td>
<td>SAMN000 43742</td>
<td>SRR063903, SRR063904</td>
<td>2820502</td>
<td>5533454</td>
<td>64.0%</td>
<td>1.33</td>
</tr>
<tr>
<td>638754422</td>
<td>Female, Healthy</td>
<td>SAMN007 5991</td>
<td>SRR061730, SRR061731</td>
<td>2944584</td>
<td>5823782</td>
<td>77.7%</td>
<td>0.96</td>
</tr>
<tr>
<td>764143897</td>
<td>Female, Healthy</td>
<td>SAMN000 71891</td>
<td>SRR063539, SRR063548, SRR063549</td>
<td>2496427</td>
<td>4945024</td>
<td>70.6%</td>
<td>1.27</td>
</tr>
<tr>
<td>604812005</td>
<td>Male, Healthy</td>
<td>SAMN000 6554</td>
<td>SRR063905, SRR063906</td>
<td>2680706</td>
<td>5287590</td>
<td>79.6%</td>
<td>0.75</td>
</tr>
<tr>
<td>763435843</td>
<td>Male, Healthy</td>
<td>SAMN000 37012</td>
<td>SRR063553, SRR063554, SRR063555</td>
<td>2513710</td>
<td>4962822</td>
<td>73.1%</td>
<td>1.11</td>
</tr>
<tr>
<td>763961826</td>
<td>Male, Healthy</td>
<td>SAMN000 40248</td>
<td>SRR063583, SRR063584, SRR063585</td>
<td>2436744</td>
<td>4798677</td>
<td>77.4%</td>
<td>0.98</td>
</tr>
</tbody>
</table>
For examining the characteristics of the read data sets, the BWA [71] aligner was used to align the sequence reads against the Human Microbiome Project microbiome reference sequences [159]. Each read was classified to a genus by its best alignment hit (Additional file 1). Table 9.1 displays the percentage of reads that could be mapped back to a reference genome for each data set. Fig. 9.4 shows the median read percentages assigned to highly abundant genera that at least 0.5% of reads were assigned to in at least three samples. This threshold was selected to eliminate low abundance genera as well as genera that were highly abundant in only one or two individuals. We also downloaded Illumina data sets sequences from the same set of healthy individuals to show that the genera distribution in the samples is consistent. Fig. 9.4 shows that the median percentage of reads for highly abundant genera is very similar between the Illumina and 454 read data sets, providing confidence that the sequence process was able to correctly capture the abundance ratios. Figure 9.4 also shows distinct differences in the abundances of major genera present in the Crohn’s and healthy individuals with statistically significant decreases in Alistipes, Bacteroides, Faecalibacterium, and Parabacteroides in Crohn’s disease samples. The genera Bifidobacterium, Blautia, Clostridium, Coprococcus, Dorea, Enterococcus, Lactobacillus, Ruminococcus, Streptococcus, and Veillonella were significantly increased in Crohn’s disease samples. The Mann–Whitney U test was used to calculate p-values. Previous studies have found a wide range of alterations in the microbiome of Crohn’s disease patients versus healthy individuals [160]. Examples of frequent shifts found previously in Crohn’s disease microbiota composition are decreases in Faecalibacterium prausnitzii, increases in Ruminococcus gnavus, and increases in Enterococcus faecium [161][162][157][163]. The consistency between Illumina and 454 data sets and observations of microbiota shifts found in previous literature provides evidence that our selected data sets provide an appropriate view of biological differences between the microbiome of healthy individuals and individuals with Crohn’s disease.
**Figure 9.4: Taxonomic read classification** Median percentage of reads assigned to major genera present in the Crohn’s disease and healthy read data sets. A ‘*’ denotes a significant difference in the median read percentages in the Crohn’s and healthy samples. 2016 BMC Genomics reprinted with permission
9.3.2 Repetitive and Transposable Elements are Associated with Node Characteristics

9.3.2.1 Shannon’s Index Score Distribution and Functional Gene Categories

This section provides an overview of the distribution of Shannon’s index scores found across the nodes in the hybrid graphs of the Crohn’s disease and healthy data sets. Fig. 9.5 displays the distribution of node counts for the Shannon’s index scores. Notice that the Shannon’s index scores that have the greatest node counts fall in the range of .6 to .7. If a node had a single in and out edge representing a single unique path and the in and out edges were evenly weighted, then its corresponding Shannon’s score would be \( \ln(2) \approx .69 \). Thus, nodes whose corresponding sequence is a unique genomic region will have Shannon’s index scores in this score range.

The SEED [153] is an organizational database system that provides five levels of hierarchical gene functional categorization with the first level being the most general level of classification. The FigFams, which form the leaves of this hierarchy, are sets of proteins that share the same function and are similar at the sequence level. FragGeneScan [164] was used to predict genes in contigs for all data sets. We downloaded the SEED protein database and used DIAMOND, which was chosen because of its scalability to large data sets and similar degree of sensitivity as BLASTX, to align the predicted genes to the SEED FigFams at a 40% identity threshold. The SEED subsystems database was used to assign each gene to a level 1 functional categorization if possible. Most of the predicted genes were located on contigs whose corresponding nodes fell into the .6-.7 score range as well, as shown by Fig. 9.5. However, there are many outlier genes that have a much greater Shannon’s index score, indicating that they might be found on contigs whose nodes represent repetitive sequence or sequence that is shared between two or more species. In the following section, we first provide a brief characterization of the most extreme outlier genes, showing that many of these genes are transposase and integrases. We then demonstrate that nodes annotated with repetitive and mobile genetic elements have a greater proportion of high Shannon index scores compared to nodes annotated with other gene categories.
Figure 9.5: Shannon’s index score distribution and Seed subsystem assignment. Node count distribution for assigned Shannon’s index scores. Boxplots display the range of scores for predicted genes assigned to level 1 classifications in the SEED subsystems. A) Shannon’s index score distribution of the nodes of the hybrid graphs generated from the Crohn’s disease datasets. B) Shannon’s index score distribution of the nodes of the hybrid graphs generated from the healthy data sets. Note that the highest node counts and predicted genes fall into the same score range for both sample types. 2014 BMC Genomics reprinted with permission.
9.3.2.2 Characterization of Biological Features on Outlier Nodes

Here we briefly examine the biological features on nodes with the most extreme Shannon’s index scores. The two highest scoring nodes in each data set that had at least one edge with minimum edge weight of 5000 were obtained from the hybrid graphs. The minimum edge weight was set to filter low coverage nodes in the data set. Blastx against the NCBI non-redundant protein database was used to identify biological features on the contigs corresponding to the selected nodes. Table 9.2 displays the results of the feature hits found on the contig sequences. The most frequent hits that were not to hypothetical or uncharacterized proteins were to transposase and integrase related elements. A total of 33.3% of the hits were to transposases and integrases.

9.3.2.3 Selection of a Threshold for High Shannon’s Index Scores

In the previous section we examined the biological features on a small subset of nodes with extreme Shannon’s index scores. Next, we demonstrate that nodes annotated with repetitive and mobile elements have a greater proportion of high Shannon’s index scores. However, minimum threshold for a Shannon’s index score to be considered high must be defined. Recall that for a given node with \( n \) edges, the maximum Shannon’s index score that can be assigned to that node is \( \ln(n) \). An appropriate threshold will exclude nodes that possess a single entering and exiting edge as these nodes might be more likely to be part of unique genomic region. The minimum threshold that would eliminate these nodes is \( \ln(2) \approx .69 \) as this is the maximum Shannon’s index score that could be assigned to a node with two edges. However, a node could possess two evenly weighted edges and a third spurious edge that has a small edge weight, pushing this node past the minimum threshold. Thus the minimum threshold is raised to \( \ln(3) \), which is the maximum score a node with three evenly weighted edges could be assigned. For the sake of simplicity \( \ln(3) \approx 1.1 \) is rounded to one.
Table 9.2: Sequence Features on Nodes with the Highest Shannon’s Indexes. 2016 BMC Genomics reprinted with permission

<table>
<thead>
<tr>
<th>Sample</th>
<th>Shannon's Index Score</th>
<th>Sequence Feature(s)</th>
<th>Blast E-Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Female 33</td>
<td>3.69</td>
<td>Transporter, RelB/DinJ, Transposase</td>
<td>5e-15, 5e-32, 4e-32</td>
</tr>
<tr>
<td>Female 33</td>
<td>3.66</td>
<td>Transposase</td>
<td>1.0e-45</td>
</tr>
<tr>
<td>Female 58</td>
<td>2.58</td>
<td>Hypothetical protein</td>
<td>6.0e-04</td>
</tr>
<tr>
<td>Female 58</td>
<td>2.46</td>
<td>TonB-dependent receptor</td>
<td>6.0e-51</td>
</tr>
<tr>
<td>Female 92</td>
<td>2.82</td>
<td>Delta-lactam-biosynthetic de-N-acetylase</td>
<td>3.0e-57</td>
</tr>
<tr>
<td>Female 92</td>
<td>2.74</td>
<td>Resolvase</td>
<td>4.0e-87</td>
</tr>
<tr>
<td>Male 104</td>
<td>2.53</td>
<td>Transposase, Cbl</td>
<td>3e-32, 7e-27</td>
</tr>
<tr>
<td>Male 104</td>
<td>2.47</td>
<td>Phosphatase, Histidine phosphotransferase</td>
<td>2e-108, 4e-72</td>
</tr>
<tr>
<td>Male 68</td>
<td>2.86</td>
<td>PG1 protein</td>
<td>1.0e-32</td>
</tr>
<tr>
<td>Male 68</td>
<td>2.43</td>
<td>Transposase</td>
<td>1.0e-45</td>
</tr>
<tr>
<td>Female 638754422</td>
<td>3.75</td>
<td>Transposase, IS4 family</td>
<td>8.0e-61</td>
</tr>
<tr>
<td>Female 638754422</td>
<td>3.68</td>
<td>Transposase</td>
<td>2.0e-22</td>
</tr>
<tr>
<td>Female 763820215</td>
<td>3.26</td>
<td>Major facilitator transporter</td>
<td>0.0e+00</td>
</tr>
<tr>
<td>Female 763820215</td>
<td>2.51</td>
<td>ATPase AAA</td>
<td>1.0e-120</td>
</tr>
<tr>
<td>Female 764042746</td>
<td>2.69</td>
<td>Transposase</td>
<td>1.0e-34</td>
</tr>
<tr>
<td>Female 764042746</td>
<td>2.64</td>
<td>DEAD/DEAH box helicase</td>
<td>1.0e-172</td>
</tr>
<tr>
<td>Female 809635352</td>
<td>4.65</td>
<td>30S ribosomal protein S12</td>
<td>1.0e-26</td>
</tr>
<tr>
<td>Female 809635352</td>
<td>4.58</td>
<td>Uracil phosphoribosyltransferase</td>
<td>7.0e-05</td>
</tr>
<tr>
<td>Female 764143897</td>
<td>3.90</td>
<td>None</td>
<td>NA</td>
</tr>
<tr>
<td>Female 764143897</td>
<td>3.63</td>
<td>None</td>
<td>NA</td>
</tr>
<tr>
<td>Male 604812005</td>
<td>2.96</td>
<td>Tetratricopeptide repeat protein</td>
<td>6.0e-47</td>
</tr>
<tr>
<td>Male 604812005</td>
<td>2.71</td>
<td>ATP-dependent DNA helicase RecQ</td>
<td>2.0e-149</td>
</tr>
<tr>
<td>Male 763435843</td>
<td>3.65</td>
<td>Putative transposase, Major Facilitator Superfamily protein, Glycosyltransferase, Group 1 family protein</td>
<td>5e-12, 1e-24, 8e-13</td>
</tr>
<tr>
<td>Male 763435843</td>
<td>3.47</td>
<td>Transposase</td>
<td>1.0e-43</td>
</tr>
<tr>
<td>Male 763961826</td>
<td>2.72</td>
<td>None</td>
<td>NA</td>
</tr>
<tr>
<td>Male 763961826</td>
<td>2.65</td>
<td>Transposase family protein, DNA polymerase IV</td>
<td>4e-67, 2e-51</td>
</tr>
</tbody>
</table>
9.3.2.4 Characterization of Biological Features on High Scoring Nodes

In this section, it is demonstrated that nodes annotated with repetitive and mobile genetic elements have a greater proportion of high Shannon’s index scores. To achieve this, we compare the proportion of nodes assigned high Shannon index scores for each of the SEED functional categories to the proportion of nodes assigned high Shannon index scores for rRNA operon and transposase/integrase sequences.

The Meta-RNA [165] software tool was used to predict rDNA operon sequences in all of our contig sets. Meta-RNA was chosen because of its ability to detect rRNA sequences in fragmented metagenomics data. To further investigate the distribution of transposase and integrase sequences across nodes, the protein sequences of all transposases and integrases were downloaded from the ACLAME database. DIAMOND was used to align the predicted genes to the transposase and integrase protein sequences from both the ACLAME library and SEED FigFams at a 40% identity threshold. For each read set in the Crohn’s disease and healthy control data sets, the proportion of nodes with Shannon’s index scores greater than one for each of the SEED functional categories, the rRNA operon sequences, and the transposase and integrase sequences was determined. The paired Wilcoxon test was applied to compare the high scoring node proportions for each SEED functional category pooled from the Crohn’s and healthy data sets against the pooled rRNA operon high scoring node proportions followed by the pooled transposase and integrase sequence high scoring node proportions. The paired Wilcoxon tests with FDR correction showed that both the transposases and integrases and rRNA operons had a significantly higher proportion of nodes with Shannon’s index scores greater than one than the SEED functional categories (q = 2.44x10^{-04}; Additional file 2 from [128]).
9.3.3 Mining and Characterization of Transposase associated Antibiotics Resistance Genes

As reviewed in the background, in addition to transposases, bacterial transposons often carry genes for antibiotics resistance allowing for the spread of antibiotic resistance mechanisms [141]. In this section, for each transposase-associated node with a Shannon’s index score greater than one, the genera of the sequences that contain that transposase are identified. For a given node, the contigs corresponding to each of the node’s adjacent neighbors in the hybrid graph are obtained. Majority read vote was used to assign each contig to a genus by the Human Microbiome Project microbiome reference sequences. If a contig could not be classified to a genus then it was classified as unknown. For each transposase-associated node, a vector \( v = (x_1, x_2, \ldots, x_n) \) was created, where \( x_i \) is the summed length of the neighboring contigs assigned to genus \( i \) normalized by the total length of all of the neighboring contigs. K-means clustering was used to cluster the high scoring transposase nodes into groups based on the Euclidean distance of these vectors, which represent the distribution of the genus level classifications of the sequences containing each transposase region. Transposase nodes that had more than 20% of adjacent sequence classified as unknown were not included in the clustering. Multiple iterations of k-means clustering and the generated elbow plots shown in Fig. 9.6 were used to select ten as the number for \( k \) for the transposase nodes from the healthy data set and twenty for the transposase nodes from the Crohn’s disease data set. For the purpose of examining the occurrence of antibacterial resistance genes among phylogenetically conserved transponsases, all of the antibacterial resistance gene protein sequences were downloaded from the CARD database. Any resistance gene tagged as a gene variant was removed from the set to avoid false positive hits. DIAMOND was used to align the predicted genes in the contigs for each data set against the antibiotic resistance gene proteins at a 90% identity threshold. For each transposase node we extracted all of the contigs from its 5-neighborhood node set to search for hits to antibacterial resistance genes localized near transposase sequences. The 5-neighborhood of a given node is the
Figure 9.6: K-means clustering elbow plots Elbow plots for the k-means clustering of the transposases nodes for the Crohn’s disease and healthy data set. The left plot A) shows the within groups sum of squares for the Crohn’s disease data sets and the right plot B) shows the sum of squares for the healthy data sets. The within sum of squares was much higher for the Crohn’s disease data sets versus the healthy data sets. The number of clusters for the Crohn’s disease data sets (10) and the healthy data sets (20) were chosen such that their sum of squares were roughly equivalent. 2016 BMC Genomics reprinted with permission.
set of nodes no further than a path distance of five from that node. Fischer’s test was used to determine if the number of hits to classes of antibiotics resistance genes in the neighborhoods of the transposase nodes was enriched in comparison to the total number of hits in the total nodes set. Figure 9.7 and 9.8 shows the phylogenetic transposase clusters for the Crohn’s disease and the healthy data sets, respectively. Each pie chart displays the average distribution of the abundant genera (at least 5% of the total composition; Additional file 3) of the contigs of the neighboring nodes of each transposase-associated node in that cluster. For each cluster we list the number of 5-neighborhood node set hits to antibacterial resistance gene classes. FDR corrected enrichments at the .05, .01, and .001 significance levels are indicated and can also be found in Additional file 4 in [128]. The number of transposase-associated nodes in each cluster are listed above each pie chart. If a transposase cluster had less than twenty members, then it was not included in Fig. 9.7 and 9.8 or subsequent analysis. Also, two clusters from the Crohn’s disease data set had redundant phylogenetic distributions of highly abundant genera; the larger cluster was used for further analysis.

In the transposase clusters generated from the Crohn’s disease data sets, there were several clusters that were enriched for antibiotic resistance gene classes. In particular, there was an Enterococcus phylogenetic transposase cluster that was not found in the healthy control data set, shown in Fig. 9.7 (e). The node set obtained from the 5-neighborhood of all of the transposase associated nodes in the Enterococcus cluster was enriched with aminoglycoside, macrolide, and streptogramin resistance gene classes. The aminoglycoside resistance gene class was enriched at the .001 significance level and represented hits to the intrinsic Enterococcus Faecium aac(6')-li gene. The macrolide and streptogramin classes were enriched at the .01 level of significance and represented hits to the intrinsic Enterococcus Faecium msrC gene. A single hit to the tetracycline resistance gene class was most similar to the tet(L) gene and aligned to a Enterococcus plasmid. Two clusters whose transposase-associated nodes had many neighbors
Figure 9.7: Phylogenetic clusters of transposases in the Crohn’s disease data sets with antibiotic class enrichments. Transposase associated nodes were clustered using k-means clustering according to the distribution of genera that the contigs of their neighboring nodes were assigned to. Enrichments of antibiotic resistance gene classes for the 5-neighborhood of the transposase nodes are indicated at the .05, .01, and .001 significance level (*, **, ***). 2016 BMC Genomics reprinted with permission.
with contigs classified to *Lactobacillus* were also significantly enriched with antibacterial resistance gene classes, shown in Fig. 9.7 (d,e).

The tetracycline class hits were most similar to tet(W) genes found in *Bifidobacterium*, *Lactobacillus*, and *Streptococcus*. The streptogramin class hits were to the vat(E) gene found in *Enterococcus Faecium* and some *Lactobacillus* plasmids. The *Ruminococcus* group shown in Fig. 9.7 (i) was enriched with tetracycline resistance genes with hits to tet(O) and tet(W).

In the healthy control data sets, resistance genes were most prevalent in transposase clusters associated with *Bacteroides* and *Prevotella*. The *Bacteroides* cluster, Fig. 9.8 (h) was also the largest cluster in the group. Fig. 9.8 (d, e, h, i, j) were all enriched resistance genes from the beta-lactam, lincosamide, macrolide, streptogramin, and tetracycline resistance gene classes. The enrichments for the lincosamide, macrolide, streptogramin resistance gene classes were due to hits to the ermG and ermF macrolide-lincosamide-streptogramin B resistance proteins. The ermB, ermF, ermG, and ermS genes are common sources of resistance in *Bacteroidales* strains found in the intestine [166]. The enrichments for the beta-lactam class of resistance genes were due to hits to class A beta-lactamases which are found in strains of *Bacteroides* and *Prevotella*. Tetracycline class enrichments were from hits to the tet(Q) resistance gene, also found in *Bacteroides* and *Prevotella*. The transposase cluster associated with *Bacteroides* and *Alistipes*, Fig. 9.8 (b), was enriched for class A beta-lactamase and tet(Q) resistance genes. Antibiotics resistance gene hits with gene descriptions can be found in Additional file 5.

### 9.3.4 Additional Features Associated with Transposase sequences

This section presents the results of examining all any additional gene features associated with transposase sequences in the Crohn's disease and healthy gut microbiome data sets. First, gene features associated with transposase sequences were extracted as described in the methods. Each gene feature was assigned a level 2 SEED subsystem classification. The counts for each
Figure 9.8: Phylogenetic clusters of transposases in the healthy data sets with antibiotic class enrichments. Transposase associated nodes were clustered using k-means clustering according to the distribution of genera that the contigs of their neighboring nodes were assigned to. Enrichments of antibiotic resistance gene classes for the 5-neighborhood of the transposase nodes are indicated at the .05, .01, and .001 significance level (*, **, ***). 2016 BMC Genomics reprinted with permission.
Subsystem were normalized per data set by dividing the subsystem count by the total number of nodes explored by neighborhood analysis and then multiplying by a thousand to get number of subsystem hits per thousand nodes. Next the Mann-Whitney U test was used to determine whether there is a significant difference in subsystem category hits for the Crohn's disease versus the healthy data sets. For subsystems that have significantly different number of hits between the Crohn's disease and healthy data sets, the distribution of bacterial genera in which the subsystems occurs was determined. Finally, three of the significant features: Mannose binding, Beta galactoside, and Heme binding in gram positives were explore in more detail in the context of their possible roles in the gut microbiome of Crohn's disease individuals and previous literature.

9.3.4.1 Subsystem Analysis

This work extends the research on antibiotic resistance genes described in the previous sections. To begin, all transposase associated gene features were assigned a level 2 subsystem classification. If a given subsystem category did not have at least five hits for at least two of the data sets it was not included in subsequent analysis. Fig. 9.9 displays the median subsystem category hits per thousand nodes for the significant subsystems. Many of these subsystems are related to metabolic functions. For example, lactate fermentation had many more hits in the Crohn’s disease data sets in comparison to the healthy data sets. In the next section, it is shown that this function is found in Lactobacillus. According to [167] metabolic genes may be preferentially transferred horizontally between bacterial types, allowing for adaptation to novel energy sources.

---

5 Material in this chapter has been previously published: J. Warnke-Sommer and H. Ali, “Graph-based Analysis of Genetic Features Associated with Mobile Elements in Crohn’s Disease and Healthy Gut Microbiomes”, BISTEC 2017, © 2017 SciTePress
Figure 9.9: Transposase associated subsystem differences between Crohn’s disease and healthy gut microbiome samples. Median gene hits to each significant subsystem are shown for Crohn’s (red) and healthy (blue) samples. © 2017 SciTePress reprinted with permission
Here a brief description of each of the subsystems that were significantly different in the Crohn’s and healthy gut microbiome data sets is given.

• Alginate metabolism: Alginate is a polysaccharide found in the cell walls of brown algae and capsules of soil bacteria [168]. Alginates are commonly added to food as a thickener or stabilizer [169].

• Beta-Glucoside metabolism: A glucosides is a glucose sugar molecule(s) that is attached to another non-sugar functional group. They are commonly found in plant material [170].

• Cellulosome: Cellulosomes are large enzymes capable of digesting cellulose, a polysaccharide that is the major component of cell walls [171].

• Chorismate synthesis: Chorismate is a precursor in the synthesis of aromatic amino acids in most prokaryotes [172].

• Conjugative transposon, Bacteroides: This is a DNA element that excises itself, forms a circular intermediate, and then reintegrates itself into the same genome or transfers between cells to a different genome [173].

• Fermentations: Lactate: In bacteria, lactate is produced by the fermentation of carbohydrates [174].

• Fructooligosaccharides (FOS) and Raffinose Utilization: FOS and Raffinose are nondigestible oligosaccharides that can be found in plants [175].

• Heme, hemin uptake and utilization systems in Gram Positives: Iron is often an enzyme cofactor in many prokaryotic biological processes. Many pathogens obtain iron through the uptake of heme [176].
• Iron acquisition in Vibrio: This includes Ton-B dependent transport of heme in Gram Negatives [177].

• Maltose and Maltodextrin Utilization: Maltose is a disaccharide that can be formed from the digestion of starch [178]. Maltodextrin is a modified starch that is commonly used as a food additive [179].

• NAD and NADH cofactor biosynthesis global: The pyridine nucleotide redox pair NAD/NADH is an essential cofactor for all living organisms [180].

• Phage integration and excision: Bacteriophages are capable of integrating their DNA into a host genome [181].

• Proteasome bacterial: These perform protein degradation in bacteria to maintain homeostasis [182].

• tRNA aminoacylation: Aminoacyl-tRNA synthetases catalyse the addition of amino acid to a transfer RNA [183].

Fig. 9.10 displays the genera distribution for each of the significant subsystems. In the healthy samples, most of the transposase-associated subsystems were found in Bacteroides. The transposase-associated subsystems that were found in the Crohn’s disease samples were more distributed across bacterial genera. For an example, the subsystem Fermentation:Lactate was found mostly in Lactobacillus. The subsystem Fructooligosaccharides (FOS) and Raffinose utilization was found commonly in Bifidobacterium. Species of Bifidobacterium are known to be capable of fermenting Fructooligosaccharides [184].

Finally, Table 9.3 displays the subsystems that were not found to be significant between the Crohn’s disease and healthy gut microbiome data sets.
Distribution of Genera for Significant Subsystems

Figure 9.10: Distribution of Genera for Significant Subsystems. (A) shows the genera distribution for subsystems with a greater number of hits in Crohn’s samples and (B) shows the genera distribution for subsystems with a greater number of hits in the healthy samples. © 2017 SciTePress reprinted with permission
### Table 9.3: Subsystems that were not Significantly Different between Crohn’s Disease and Healthy Microbiome Samples.

© 2017 SciTePress reprinted with permission

<table>
<thead>
<tr>
<th>Subsystem</th>
<th>Pathway</th>
</tr>
</thead>
<tbody>
<tr>
<td>16S rRNA modification within P site of ribosome</td>
<td>Entner-Doudoroff Pathway</td>
</tr>
<tr>
<td>5-FCL-like protein</td>
<td>Folate Biosynthesis</td>
</tr>
<tr>
<td>ABC transporter oligopeptide (TC 3.A.1.5.1)</td>
<td>Galactosylceramide and Sulfatide metabolism</td>
</tr>
<tr>
<td>Alanine biosynthesis</td>
<td>Glycogen metabolism</td>
</tr>
<tr>
<td>Alkanesulfonate assimilation</td>
<td>Group II intron-associated genes</td>
</tr>
<tr>
<td>Ammonia assimilation</td>
<td>Heat shock dnaK gene cluster extended</td>
</tr>
<tr>
<td>Aromatic conversions and predicted Co2 transporter cluster</td>
<td>High affinity phosphate transporter and control of PHO regulon</td>
</tr>
<tr>
<td>At1g69340 At2g40600</td>
<td>Histidin Biosynthesis</td>
</tr>
<tr>
<td>At3g21300</td>
<td>L-fucose utilization</td>
</tr>
<tr>
<td>At5g63420</td>
<td>L-rhamnose utilization</td>
</tr>
<tr>
<td>ATP-dependent Nuclease</td>
<td>Lactose and Galactose Uptake and Utilization</td>
</tr>
<tr>
<td>Bacterial Cell Division</td>
<td>Mannose Metabolism</td>
</tr>
<tr>
<td>Bacterial Cytoskeleton</td>
<td>Methionine Biosynthesis</td>
</tr>
<tr>
<td>Beta-lactamase</td>
<td>Multidrug Resistance Efflux Pumps</td>
</tr>
<tr>
<td>Beta-lactamase cluster in Streptococcus</td>
<td>Oxidative stress</td>
</tr>
<tr>
<td>C. jejuni colonization of chick caeca</td>
<td>Potassium homeostasis</td>
</tr>
<tr>
<td>Calvin-Benson cycle</td>
<td>Purine conversions</td>
</tr>
<tr>
<td>Cell Division Subsystem including YidCD</td>
<td>Restriction-Modification System</td>
</tr>
<tr>
<td>Cell division-ribosomal stress proteins cluster</td>
<td>Ribonucleotide reduction</td>
</tr>
<tr>
<td>Chitin and N-acetylglucosamine utilization</td>
<td>Ribosome LSU bacterial</td>
</tr>
<tr>
<td>D-Galacturonate and D-Glucuronate Utilization</td>
<td>Sialic Acid Metabolism</td>
</tr>
<tr>
<td>De Novo Pyrimidine Synthesis</td>
<td>Streptococcal group antigen operons</td>
</tr>
<tr>
<td>DMT transporter</td>
<td>Tetracycline resistance, ribosome protection type</td>
</tr>
<tr>
<td>DNA Repair Base Excision</td>
<td>Transcription factors bacterial</td>
</tr>
<tr>
<td>DNA repair, bacterial</td>
<td>tRNA modification Archaea</td>
</tr>
<tr>
<td>DNA repair, bacterial RecFOR pathway</td>
<td>Universal GTPases</td>
</tr>
</tbody>
</table>
In-Depth Analysis of Pathologically Relevant Subsystems

In this section, an in-depth summary of the Beta-Glucoside, Maltose and Maltodextrin Utilization, and Heme, Hemin Uptake and Utilization subsystems are presented.

According to Fig. 5 beta-glucoside subsystem functions are found most commonly in *Ruminococcus, Clostridium*, and *Lactobacillus* genera. A large portion of the beta-glucoside functions occurs in unknown genera. Glucosidases are capable of producing metabolites that are implicated in colon cancer [185]. Crohn’s disease is associated with higher risk of colorectal cancer [186].

The subsystem Heme, Hemin, Uptake and Utilization are found most commonly in *Streptococcus* and *Lactobacillus* as well as in unknown genera. Iron is a cofactor of many enzymes in living systems. Its uptake is essential for pathogenic infection [176]. In mammalian systems, the most abundant form of iron is bound in heme. Thus many bacterial systems have developed for the uptake of heme. The availability of host heme to bacterial pathogens can greatly increase the difficulty of clearing an infection [187].

Finally, the subsystem Maltose and Maltodextrin is found in *Bifidobacterium, Lactobacillus, Collinsella*, and *Enterococcus*. Some studies have found that the food additive Maltodextrin can alter the bacterial homeostasis of the intestine [188]. Maltodextrin has also been found to increase adhesion of Crohn’s disease associated *E. coli* [179] and promote *Salmonella* mucosal colonization and survival [189].

Fig. 9.11 provides a more granular view of the level 3 subsystems classification counts for Beta-Glucoside Metabolism, Heme, Hemin Uptake and Utilization, and Maltose and Maltodextrin Utilization.
**Figure 9.11:** Level 3 subsystem counts for Beta-Glucoside Metabolism, Heme, Hemin Uptake and Utilization Systems, and Maltose and Maltodextrin Utilization. © 2017 SciTePress reprinted with permission
9.4 Conclusion and Discussion

We have developed a novel graph mining and assembly algorithm that is capable of extracting useful biological information and producing high quality assembly results. This algorithm captures genome structural information using a hybrid graph. The initial overlap graph is incrementally reduced using heavy edge matching and node merging to create a graph spectrum, $G_0, G_1, \ldots G_n$ that represents a read data set at multiple levels of information. To provide the most accurate yet succinct representation of the input data set, nodes from each graph level are selected as best representatives of their corresponding read clusters and combined into a single hybrid graph $G'_0$. Each node in this graph represents either a unique region, repetitive element, or region conserved between multiple species. A Shannon’s index score is assigned to each node to numerically describe the number of incident edges and the evenness of their weights. It is shown that repetitive elements, in particular rRNA operons and transposase genes, are associated with higher Shannon’s index scores. We then extract transposase genes whose corresponding nodes had high Shannon’s index scores in five read data sets obtained from the gut microbiome of individuals with Crohn’s disease and eight read data sets obtained from the gut microbiome of healthy controls. The resulting transposase genes were clustered into groups determined by the distribution of genera that the contigs obtained from the adjacent neighbors of their corresponding nodes were classified too. We then test for enrichment of antibiotic resistance genes in the 5-neighborhood of the nodes in each transposase cluster. Distinct differences were apparent in the Crohn’s disease and control data set clustering results. An enterococcal transposase cluster that was enriched with various antibacterial resistance gene classes was present in the Crohn’s disease clustering results while being absent from the healthy control clustering results. *Enterococcus* species are often overrepresented in Crohn’s disease data sets. Other sources of antibacterial resistance genes were from *Lactobacillus* associated transposase clusters. Origins of antibiotic resistance in healthy individuals were heavily biased towards *Bacteroidales* species. The distribution of the number of transposases was relatively even across
the Crohn’s disease clusters, while in the healthy disease data sets most transposases were found in a *Bacteroides* associated cluster.

In aim 3, results demonstrate several important transposase associated genetic features that are more prevalent in Crohn’s disease gut microbiome samples than healthy samples. Several of these functions have been implicated in previous research as biologically relevant to Crohn’s disease and associated conditions. These results provide insights into gene features that may allow gut bacteria to adapt to their ecological niches in healthy and disease states. In conclusion, this chapter highlights the ability of the assembly graph to be a powerful data-mining support that can capture meaningful biological information and patterns in its structural features.
CHAPTER 10

A TOLERANCE GRAPH APPROACH FOR DOMAIN SPECIFIC ASSEMBLY OF NEXT GENERATION SEQUENCING DATA

10.1 Introduction

In this chapter, the customization of the assembly process based on input data set characteristics is explored. The assembly problem is a computationally non-trivial task. Besides the extremely large size of the datasets, genome features such as repeat content and sequencing features such as sequencing error introduce false-positive edges into the assembly graph [190]. Furthermore, average read coverage might vary for different regions of the target sequence dependent on sequence composition [191] and other various factors leading to assembly gaps.

For the purpose of eliminating false-positive edges in the assembly graph that would confound the assembly process while retaining maximal information, assemblers utilize graph parameters to control the stringency of the assembler. Overlap graph based assemblers often specify a minimum overlap length parameter and minimum overlap alignment identity parameter for each edge to control the assembly process. The major parameter that is used to control the assembly stringency of De Bruijn graph based assemblers is the length of the k-mer. Multiple studies have demonstrated that these parameters can dramatically impact the quality of assembly results for both De Bruijn and overlap graph assemblers [192].

The optimal configuration of graph parameters will most likely not be the same for datasets that have differing feature characteristics. For the purpose of building a knowledge-based approach for the assembly process, it is important to identify assembler performance characteristics for specific assembly domains. Furthermore, next generation sequencing dataset characteristics such as coverage and underlying genome complexity can greatly vary within the same dataset. Most current assembly algorithms do not customize graph parameters locally to reflect intra-dataset dynamics but instead apply a single parameter to the entire overlap graph or
iterate through a series of parameter values. The majority of these assemblers were developed to fill an important gap; however, they were developed with a pure computational focus without taking the properties of the input datasets into consideration. Generic assemblers that are data independent are unlikely to produce accurate solutions in all problem domains. The ability to incorporate domain-specific and data specific information into the assembly process would be beneficial.

Here, we build a knowledge base for the Focus assembly method by evaluating its performance when applied to datasets from bacterial genomes that differ in domain characteristics. In particular, we study the effects of genome composition and coverage depth on assembly quality. Furthermore, these datasets are assembled across the parameter space spectrum for minimum fragment overlap length and minimum fragment overlap alignment identity for the purpose of characterizing optimal parameter configurations as a function of dataset characteristics. Finally, we introduce a new graph model for data-dependent assembly called the tolerance graph. In this approach, signals associated with local features in the input dataset as extracted. These signals are utilized to locally customize graph parameters as a linear weighted sum of scores of local read and graph specific characteristics and a user-provided threshold. This data-dependent information will allow for an intelligent, knowledge-based approach to the assembly process.

10.2 Methods

10.2.1 Graph Theoretic Model

The interval overlap graph has been an important tool for modeling the assembly process. As previously reviewed in chapter four, a graph \( G (V, E) \) is an interval graph, if each vertex \( v \in V (G) \) corresponds to an interval \( I (v) \) of the real line. An edge \( \{u, v\} \in E (G) \) exists if and only if \( I (u) \cap I (v) \neq \emptyset \). Reads can be envisioned as intervals of the real line. An interval overlap graph approach for assembly maps each read to a vertex in the overlap graph. Two

---

7Material in this chapter has been previously published: J. Warnke and H. Ali, “A Tolerance Graph Approach for Domain-Specific Assembly of Next Generation Sequencing Data“, 2013 IEEE 13th International Conference on, Data Mining Workshops (ICDMW), © 2013 IEEE
vertexes are adjacent to one another if and only if their corresponding reads overlap. The overlap graph and interval graph are shown in Fig. 10.1.

As discussed previously, the overlap structure of the read dataset would form an interval graph in an ideal situation. However, in real world applications this is unlikely. Sequencing error and genome complexity such as repeat regions within the genome create overlap ambiguity, leading to shortened contig lengths and mis-assembly. Furthermore, short overlap relationships may not be truly indicative that two reads are consecutive within the genome. To address the issue of overlap ambiguity, assemblers institute overlap thresholds such as minimum overlap length and minimum overlap alignment identity. It is demonstrated in this chapter that the characteristics of input datasets, such as read coverage and genome composition, influence the range of threshold values where optimal assembly results can be obtained. Results demonstrate that assembly graph parameters should not be viewed as static thresholds, but must dynamically adapt to input sequence data characteristics.

The tolerance graph provides a parameter-centric view of assembly process that reflects the dynamic nature of sequence assembly. A graph $G$ is a tolerance graph, if each vertex $v \in V(G)$ corresponds to an interval $I(v)$ of the real line and a tolerance $t_v \in R^+$. An edge $\{u, v\} \in E(G)$ exists if and only if $I(u) \cap I(v) \geq \min \{t_u, t_v\}$ [194]. In this research, the concept of a customized assembly approach by assigning unique tolerance thresholds to each node in the overlap graph is explored. These thresholds are generated by a linear sum of scores of local read and graph specific characteristics and a user-provided value.

### 10.2.2 Tolerance Threshold Adjustment

The tolerance graph provides a parameter-centric view of assembly process that reflects the dynamic nature of sequence assembly. Domain-specific features of the read dataset and the overlap graph are extracted and reintegrated into the assembly process by calculating dynamic tolerance thresholds specific to each node in the overlap graph.
Figure 10.1: **Graph theory.** (a) DNA fragments and overlap graph. (b) Interval representation and interval graph. Weights are assigned to edges. © 2013 IEEE reprinted with permission
For each node, a tolerance is calculated as a weighted score dependant on three domain-specific parameters. The first parameter is the degree of the node in the original overlap graph. The second parameter is the length of the read corresponding to the node in the overlap graph. The third parameter is a user-provided threshold in base-pairs corresponding to the minimum desired shared overlap between reads.

For the purpose of normalizing each parameter, we subtract the current parameter from the parameter average of the dataset and divide the result by the parameter standard deviation. The normalized scores and final weighted node tolerance can be calculated by formulas 1-4.

\[
Z_{score\_degree}(u) = \frac{(node\_degree(u) - average\_node\_degree)}{node\_degree\_standard\_deviation} \tag{1}
\]

\[
Z_{score\_read\_length}(u) = \frac{(read\_length(u) - average\_read\_length)}{read\_length\_standard\_distribution} \tag{2}
\]

\[
Z_{score\_minoverlap}(u) = \frac{(minoverlap\_length - average\_overlap\_length)}{overlap\_length\_standard\_deviation} \tag{3}
\]

\[
tolerance(u) = a(Z_{score\_degree}(u)) + b(Z_{score\_read\_length}(u)) + c(Z_{score\_minoverlap}(u)) \tag{4}
\]

Notice that if the values of the parameters are less than the dataset average, then the normalized score will be negative. If the value of the parameter is greater than average, then the resulting score will be positive. Thus nodes corresponding to reads that have a read length longer than the average read length of the dataset will have a higher read length parameter score, increasing the tolerance of the node assigned to that read. Nodes that have a node degree higher than the average node degree of the dataset may represent reads that are a part of a repeat segment.
of the genome or region of high coverage. Appropriately, the resulting score of the node degree parameter will be positive, increasing the tolerance of the node. Finally, if the minimum desired overlap provided by user is greater than the average overlap length shared between reads in the overlap graph, than the resulting minimum overlap parameter will be positive, increasing the tolerance of the node.

After the tolerances for each node are calculated, the overlap length for each edge \( e = (u, v) \) in the overlap graph is normalized by equation 5 shown below, for the purpose of making the overlap length comparable to the generated tolerance threshold score.

\[
Z_{\text{score\_overlap}}(u,v) = \frac{\text{overlap\_length}(u,v) - \text{average\_overlap\_length}}{\text{overlap\_length\_standard\_deviation}}
\]

(5)

If \( Z_{\text{score\_overlap}}(u,v) \geq \min\{\text{tolerance}(u), \text{tolerance}(v)\} \), than edge \((u,v)\) is kept in the overlap graph. If the edge \((u,v)\) does not meet this minimum tolerance requirement, then it is removed from the overlap graph. These assigned tolerances allow the overlap graph to be dynamically adjusted in a node by node basis to reflect localized domain-specific features including possible repeat content, coverage differences, and read length differences.

10.3 Results

In this section, we present the results from the evaluation of the proposed assembler when applied to 454 pyrosequencing datasets from bacterial genomes that differ in domain-specific characteristics, including coverage depth and genome composition. We conduct three separate experiments to determine the viability of our approach, to examine its performance in different domains, and to characterize optimal parameter configurations as a function of assembly domain. First, we present the results of applying our assembler on datasets that vary in coverage depth.
Second, we present a comparative study against an existing assembler called Mira [131] to evaluate our assembly method’s competitiveness. Finally, we apply our tolerance customization process to a 454-pyrosequencing dataset generated from *Escherichia coli*.

### 10.3.1 Experimental Design

Graph parameters such as the minimum overlap length and alignment identity have an effect on assembly quality. Capturing optimal parameter values is key to producing a quality assembly. However, the optimal configuration of assembly parameters will most likely not be the same for datasets that have differing feature characteristics. We assembled each dataset across the parameter space spectrum for minimum fragment overlap length and minimum fragment overlap alignment identity for the purpose of characterizing optimal parameter configurations as a function of dataset characteristics.

For each dataset, the minimum overlap parameter was initially set at 30 bps and was increased with a 20 bp step size until 290 bps. At each iteration step of the minimum overlap parameter, three assemblies were conducted at 90%, 93%, and 96% overlap alignment identity thresholds. Thus, a total of 42 assembly configurations were applied to each dataset.

All other assembly parameters remained constant throughout the assembly process. The minimum overlap alignment identity needed to cluster containment fragments to representative sequences was 94%. The cutoff length for dead end path trimming and bubble popping the overlap graph was set to a length of five. The minimum contig length was set to 100 bps.

Three metrics were used to evaluate assembly quality. First the N50 statistic was calculated for all assemblies. N50 is the length N such that 50% of the assembly is contained in contigs that are $\geq N$. Second, the MUMmer package [195] was used to map the contigs back to reference genomes. A contig was counted as correct if it aligned to the reference with an alignment identity greater than 98% and if the alignment covered at least 95% of the contig. Third, the total number of contigs produced by each assembly was tabulated.
10.3.2 Coverage Depth and Assembly

In this study we focus on the effects of coverage depth on assembly quality across a spectrum of parameter values. *Escherichia coli* W datasets SRR060737 and SRR060736 were downloaded from NCBI’s sequence read archive (SRA) [158]. These two datasets were combined and a Perl script was used to randomly select subsets of the combined set at 5x, 10x, 15x, and 20x coverage. The properties of these datasets can be found in table 10.1.

The complete genome sequence of *Escherichia coli* W (Genbank: CP002185.1), *Escherichia coli* W plasmid pRK2 (Genbank: CP002187.1), and *Escherichia coli* W plasmid pRK1 (Genbank: CP002186.1) were downloaded from NCBI to be used as reference sequences.

Assemblies were conducted across the spectrum of minimum overlap length and minimum overlap identity parameters for the *Escherichia coli* W datasets at 5x, 10x, 15x and 20x genome coverage. The N50 statistic was calculated for each separate assembly. We mapped each assembly back to the reference sequences and calculated the percentage of contigs that aligned to the reference sequences with an alignment identity of at least 98% and such that the alignment covered at least 95% of the mapped contig. The number of contigs produced by each assembly was also calculated.

Results shown in Fig. 10.2(a) demonstrate the assembler’s ability to adapt to different coverage depths. As coverage increases the assembler is able to utilize more information from the fragments, resulting in longer contig lengths. Furthermore, the N50 length value peaks at different minimum overlap lengths for datasets with different genome coverage. The N50 of datasets with lower coverage rates tended to peak at smaller minimum overlap lengths versus datasets with higher coverage.

The results from mapping each assembly back to the reference sequences shown in Fig. 10.2(a), 10.2(b) demonstrate that there is a slight inverse relationship between the N50 length of the assembly and the number of contigs that align to the reference genome and meet alignment
Table 10.1: *Escherichia Coli* Data Set Characteristics

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Coverage depth</th>
<th>Total Fragments</th>
<th>Avg. fragment length (bps)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>5x</td>
<td>56 631</td>
<td>425</td>
</tr>
<tr>
<td>2</td>
<td>10x</td>
<td>113 769</td>
<td>425</td>
</tr>
<tr>
<td>3</td>
<td>15x</td>
<td>174 876</td>
<td>425</td>
</tr>
<tr>
<td>4</td>
<td>20x</td>
<td>232 248</td>
<td>425</td>
</tr>
</tbody>
</table>
Figure 10.2: Coverage depth and assembly. (A) Assembly N50 statistic. (B) Fraction of contigs that aligned to the reference sequences with an identity greater than 98% and at least 95% alignment coverage. (C) Total contigs produced by each assembly. ● = 90% overlap identity. ■ = 93% overlap identity. ▲ = 96% overlap identity. © 2013 IEEE reprinted with permission
threshold requirements. However, the percentage of contigs that can be aligned to the references sequences never drops below 88% in all cases.

Fig. 10.2(a) and Fig. 10.2(c) indicate the importance of the minimum overlap parameter. Less stringent minimum overlap parameters may introduce repeat induced false-positive edges into the overlap graph. This will cause tangles and branch points to occur in the overlap graph, resulting in reduced N50 lengths and a greater number of short, truncated contigs. In contrast, if the minimum overlap length parameter is set too stringently, many true overlap relationships between reads will not meet minimum overlap requirements. This results in information loss and gaps in the overlap graph, also reducing N50 lengths and producing a greater number of short, truncated contigs. Clearly, it is critical to optimize assembly parameters for the purpose of obtaining the best assembly results.

For the purpose of characterizing the dependency of optimal minimum overlap lengths that produce the longest N50 lengths on coverage depth. We plotted the value of the minimum overlap length that produced the greatest N50 length at each coverage depth. Fig. 10.3 clearly demonstrates a linear relationship between the coverage depth and optimal minimum overlap length configurations that produce assemblies with the longest N50 lengths. This indicates that optimal parameters settings are dependent on input dataset characteristics such as sequencing coverage and should be flexible enough to take these characteristics into consideration during the assembly process.

10.3.3 Tolerance Adjustments

In this section we focus on the dynamic adjustment of node tolerances and its impact on assembly quality. This is accomplished by iterating through a series of parameter weightings and the resulting tolerances to the overlap as shown by table 10.2.
Figure 10.3: Minimum overlap lengths that produce the greatest N50 lengths at different coverage depths. © 2013 IEEE reprinted with permission
Table 10.2: Tolerance Assembly Results. © 2013 IEEE reprinted with permission

<table>
<thead>
<tr>
<th>Overlap Weight</th>
<th>Read Length Weight</th>
<th>Node Degree Weight</th>
<th>N50 (bps) 100 bps</th>
<th>N50 (bps) 1000 bps</th>
<th>Contigs &gt; 100 bps</th>
<th>Contigs &gt; 1000 bps</th>
<th># Mapped to Ref. &gt; 100 bps</th>
<th># Mapped to Ref. &gt; 1000 bps</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.1</td>
<td>0.1</td>
<td>0.8</td>
<td>12 979</td>
<td>15 364</td>
<td>5198</td>
<td>1114</td>
<td>1083</td>
<td>466</td>
</tr>
<tr>
<td>0.1</td>
<td>0.3</td>
<td>0.6</td>
<td>13 731</td>
<td>16 723</td>
<td>4982</td>
<td>1042</td>
<td>1003</td>
<td>422</td>
</tr>
<tr>
<td>0.1</td>
<td>0.5</td>
<td>0.4</td>
<td>13 445</td>
<td>16 553</td>
<td>5060</td>
<td>1057</td>
<td>1028</td>
<td>432</td>
</tr>
<tr>
<td>0.3</td>
<td>0.1</td>
<td>0.6</td>
<td>12 828</td>
<td>15 835</td>
<td>5176</td>
<td>1096</td>
<td>1086</td>
<td>455</td>
</tr>
<tr>
<td>0.3</td>
<td>0.3</td>
<td>0.4</td>
<td>16 809</td>
<td>20 012</td>
<td>4566</td>
<td>910</td>
<td>847</td>
<td>356</td>
</tr>
<tr>
<td>0.3</td>
<td>0.5</td>
<td>0.2</td>
<td>16 471</td>
<td>18 935</td>
<td>4581</td>
<td>897</td>
<td>846</td>
<td>355</td>
</tr>
<tr>
<td>0.5</td>
<td>0.1</td>
<td>0.4</td>
<td>12 220</td>
<td>15 429</td>
<td>5349</td>
<td>1094</td>
<td>1102</td>
<td>449</td>
</tr>
<tr>
<td>0.5</td>
<td>0.3</td>
<td>0.2</td>
<td>15 998</td>
<td>18 791</td>
<td>4638</td>
<td>936</td>
<td>843</td>
<td>364</td>
</tr>
<tr>
<td>0.5</td>
<td>0.5</td>
<td>0</td>
<td>15 464</td>
<td>18 055</td>
<td>4553</td>
<td>925</td>
<td>834</td>
<td>354</td>
</tr>
</tbody>
</table>
The dynamically adjusted algorithm is applied to a read dataset simulated with MetaSim from the *Escherichia coli* O157:H7 str. Sakai chromosome downloaded from NCBI (accession number NC_002695.1). The resulting dataset contained 274,923 reads with an average length of 417.2 bps.

The N50 statistic is calculated for each separate dataset for both a minimum contig length of 100 and 1000 bps. Each assembly was mapped back to the *E.coli* reference genome and calculated the number of contigs that aligned to the reference genome with an alignment identity of at least 98% and alignment coverage of at least 95%. We did this for a minimum contig length of 100 and 1000 bps. Finally the number of contigs produced by each assembly was calculated for minimum contig lengths of 100 and 1000 bps. The results are shown in table 10.2.

### 10.4 Conclusion and Discussion

In this chapter, a tolerance graph model that is used to capture the overlap relationships among the input DNA fragments was introduced. This approach is designed to be flexible enough to take the input characteristics into consideration during the assembly process. Unlike most previous assembly approaches, this knowledge-based method extracts signals from the input dataset, which are then reincorporated into the assembly process through customized tolerance graph parameters. The performance of the assembler has been evaluated on various datasets that demonstrate different domain-specific properties. Various datasets from *Escherichia coli* were assembled at different coverage depths. Results demonstrated that the assembler was able to incorporate more information into the assembly process at greater assembly coverage, resulting in longer N50 lengths. Assembly results also demonstrate a linear relationship between coverage depth and minimum overlap lengths that produced the greatest N50 lengths. Assembly of bacterial genomes that differed in genome composition suggests that the performance characteristics of the assembler are partially dependent on the composition of the assembled genome. Results from this study provide insights into the Focus assembly method’s performance
characteristics and optimal graph parameter configurations according to certain dataset characteristics. This knowledge base will allow for the assembler to be applied in an intelligent and customized manner to a wide variety of datasets, resulting in better assembly results. Furthermore, assembly will become a more efficient process as knowledge from previous assemblies and local input dataset features can be utilized to immediately determine optimal parameter values for similar assembly datasets rather than starting anew, saving both time and computational resources.
CHAPTER 11

PARALLEL FOCUS ASSEMBLER: DISTRIBUTED ASSEMBLY GRAPH

11.1 Introduction

In this chapter, the concept of domain-specific algorithm design is extended to a distributed implementation of the assembly graph model of Focus. This distributed assembly graph supports several parallel graph trimming and traversal algorithms.

Due the massive size of next generation sequencing data sets, the assembly graph can become extremely large and difficult to process. To address this issue, several parallel assemblers have been developed that distribute the assembly graph across processors. De Bruijn graph assemblers such as AbySS [106], Ray [107], PASHA [196], and the SWAP-Assembler [197] distribute the de Bruijn graph by assigning the generated k-mers to different processors. MPI communication is then used for parallel graph simplification and traversal of the distributed de Bruijn graph. Map reduce has been applied to process a string graph assembly model [198].

The parallel assemblers discussed in the previous paragraph do not take the input characteristics of the read data into consideration during the graph partitioning process. In many applications, high performance computing is primarily viewed as a means for speeding up computational solutions for the purpose of producing faster results for downstream analysis. However, we propose that high performance computing is not just a method for obtaining faster results but can be applied in intelligent ways to extract meaningful information from big data. It is very tempting to use naïve methods for high performance computing but by using domain knowledge one may get rewarded.

The contributions of this research are three-fold. 1) We show that the integration of previous knowledge of the structure of the input data set can benefit the graph partitioning algorithm's performance. Due to the linear nature of the DNA sequence, groups of nodes
representing contiguous sequence regions can simply be assigned to the same graph partition with minimal processing. Results show that this dramatically improves runtime for all data sets and the edge cut of the partitioning in the majority of cases. 2) We develop several parallel graph algorithms applied on the distributed graph model for sequence assembly. Results demonstrate greatly improved speedup times for the distributed algorithms, which include graph cleaning and graph traversal methods. 3) Finally, it is demonstrated that we are rewarded for integrating domain knowledge into the graph partitioning process. Insights into metagenomics community structure can be obtained from the resulting graph partitioning. For metagenomics read data sets, where the underlying DNA sample is obtained from a community of organisms, it is shown that related species tend to cluster within the same graph partition. This information may be useful to researchers trying to identify the composition of a given metagenomics data set.

Knowledge integration into naive algorithms may have important extensions to additional methods in high performance computing. High performance computing methods are powerful for not only scaling algorithms to big data problems, but can applied in intelligent ways to extract meaningful information from big data.

11.2 Methods

11.2.1 Graph Coarsening

The Focus algorithm follows the approach introduced by Karypis and Kumar [130] for partitioning large graphs. This multilevel approach first coarsens an input graph to form a multilevel graph set \( G = \{G_0, G_1, \ldots, G_n\} \), where \(|N(G_n)| \leq |N(G_{n-1})| \leq \ldots \leq |N(G_0)|\). The most reduced graph \( G_n \) is then partitioned into two approximately equal halves while attempting to minimize the edge-cut between the partitions. The final step is uncoarsening. During this stage the partition on \( G_n \) is projected recursively through graph levels \( G_{n-1} \) to \( G_0 \). At each projection onto a given graph \( G_i \) in the spectrum, a refinement algorithm refines the projection.

---

8Material in this chapter has been previously published: J. Warnke-Sommer and H. Ali, “Parallel NGS Assembly Using Distributed Assembly Graphs Enriched with Biological Knowledge”, 2017 IEEE 13th International Parallel and Distributed Processing Symposium Workshops (IPDPSW), © 2017 IEEE
The approach described above requires that the reduced graph be uncoarsened completely to obtain a partitioning on the original graph $G_0$. Refinement algorithms can become fairly costly for finer graph levels that have a large number of edges and nodes. However, if the data that a given graph represents have intrinsic characteristics specific to the data domain, then that domain specific knowledge can be used to improve the partitioning process. For example, genomic sequences are linear or circular in nature. Since the nodes of the overlap graph represent consecutive regions in the genome, it is likely that many groups of nodes representing the same genomic region can be safely assigned to the same partition without dramatically impacting the overall edge-cut between partitions.

The multilevel graph spectrum represents complete uncoarsening. In contrast, the hybrid spectrum represents a compromise between complete uncoarsening and incorporation of biological knowledge. Recall nodes in hybrid graph represent clusters of nodes in the original overlap graph. If there is biological knowledge regarding which nodes should cluster together then there is no need to completely uncoarsen the graph. The representative nodes in the hybrid graph represent clusters of nodes whose corresponding read clusters have already been determined to form a contiguous contig. It would be reasonable to assume that a cluster of nodes in the original overlap graph that maps to a single node in the hybrid graph would belong to the same partition.

This research demonstrates that a partitioning on the hybrid graph will be as good as of a partitioning on the overlap graph in terms of edge cut and has a much faster running time. This partitioning found on the hybrid graph can then be simply mapped to the original overlap graph if needed.

11.2.2 Graph Partitioning

The graph partitioning process proceeds as following on a coarsened graph set to create $k$ partitions. A partitioning $P$ on a graph $G$ is defined as $P = P_1, P_2 \ldots P_k$, where $P_1, P_2 \ldots P_k$ are
disjount sets of nodes of in V(G) and \( P_1 \cup P_2 \cup \ldots \cup P_k = V(G) \). Each \( P_1, P_2 \ldots P_k \) is a partition of V(G). First, an initial partitioning with two equal halves is created on the most reduced graph, \( G_n \), by a greedy graph growing algorithm. This initial partitioning is refined with the Kernighan-Lin [199] refinement algorithm. The refined partitioning is projected onto the next graph level \( G_{n-1} \). This projected partitioning is then refined by the Kernighan-Lin algorithm. The projection and subsequent refinement of the partitioning is continued until the final graph \( G_0 \) is partitioned. Each generated partition is recursively partitioned, projected, and refined until \( G_0 \) is partitioned into the desired number of sections, \( k \). After the \( k \) partitions on the multilevel graph set \( G_0, G_1 \ldots G_n \) are obtained, each graph level is refined by a heuristic \( k \)-way Kernighan-Lin algorithm. The graph partitioning method is discussed in detail in the following sections.

11.2.2.1 Greedy Graph Growing

The approach by Karypis and Kumar [130] is followed with some modifications to customize the greedy graph growing algorithm to the data characteristics. This algorithm attempts to greedily add nodes to initial partitions \( P_1 \) and \( P_2 \) on the most reduced graph \( G_n \). Define the gain \( g_{v,z} \) of a node \( v_z \) in graph \( G_n \) by placing \( v_z \) into a partition \( P_i \) as:

\[
g_{v_zP_i} = \sum_{e(v_z,v) \in E(G_n), v \notin P_i} w(v_z,v) - \sum_{e(v_z,v) \in E(G_n), v \in P_i} w(v_z,v)
\]

where, \( w(v_z,v) \) is the weight of the edge \( e(v_z,v) \). The greedy algorithm begins by choosing a random seed node \( v_s \) to insert into the partition \( P_1 \). The gains of the nodes adjacent to \( v_s \) are calculated and the nodes placed into a priority queue according to their gains. The nodes whose gains have been computed are the horizon of the currently growing partition. The node with the greatest gain \( v_g \) is removed from the priority queue and added to \( P_1 \). The gains of the neighbors
of $v_e$ are calculated and the nodes are added to the priority queue or, if they are already in the queue, their gains are updated. An example of greedy graph growing can be found in Fig. 11.1.

The growing graph algorithm in this paper attempts to maintain balanced node weight and edge weight across the partitions. The edge weight of a given partition $P_i$ is defined as the following.

$$e_{w \text{ partition}}(P_i) = \sum_{e(v_1, v_2) \in v_1 \in P, v_2 \in P} w(v_1, v_2)$$

where $w(v_1, v_2)$ is the weight of the edge $e(v_1, v_2)$. If $e_{w \text{ partition}}(P_1)$ becomes greater than $1.03e_{w \text{ partition}}(P_2)$, then the partition growing of $P_1$ is terminated. A new seed $v_s$ is chosen and added to $P_2$. The partition $P_2$ is grown according to the method introduced previously. If $e_{w \text{ partition}}(P_2)$ becomes greater than $1.03e_{w \text{ partition}}(P_1)$, then the partition growing of $P_2$ is terminated and a new seed is chosen for $P_1$. This alternating partition growing is continued until $n_{w \text{ partition}}(P_1) \geq 0.5n_{w \text{ graph}}(G_n)$ or $n_{w \text{ partition}}(P_2) \geq 0.5n_{w \text{ graph}}(G_n)$. The terms $n_{w \text{ partition}}$ and $n_{w \text{ graph}}$ are the node weight of a given partition $P_i$ or graph $G_n$, respectively, and are defined as following.

$$n_{w \text{ partition}}(P_i) = \sum_{v \in P_i} n_w(v)$$

$$n_{w \text{ graph}}(G_i) = \sum_{v \in N(G_i)} n_w(v)$$

Any remaining nodes are added to the partition with the least node weight. Thus the greedy growing algorithm will produce an initial partitioning $P = P_1, P_2$, while attempting to balance the partitions according to both node and edge weight.
Figure 11.1: Greedy graph growing. (A) Node A is placed into partition P₁. The gains of A’s adjacent neighbors are calculated. (B). The node B has the greatest gain and is placed into P₁. The gains are updated. (C). The final node C is added to P₁ according to its gain. The partitions are now evenly weighted so the greedy graph growing is terminated. © 2017 IEEE reprinted with permission
11.2.2.2 Kernighan Lin Refinement Algorithm

Once the initial partitioning $P = P_1, P_2$ is created by the greedy graph growing algorithm, it is then refined by the Kernighan Lin refinement algorithm. The Kernighan Lin refinement algorithm relies on node swapping to improve the edge cut of a partitioning. First we provide definitions needed for the Kernighan Lin algorithm. Given a node $v_z$ in a given partition $P_i$ on $G_n$, the external cost $E_{v_z}$ and the internal cost $I_{v_z}$ of $v_z$ is defined as follows.

$$E_{v_z} = \sum_{e(v_z, v) \in E} w(v_z, v)$$

$$I_{v_z} = \sum_{e(v_z, v) \in E} w(v_z, v)$$

The $D$ value of $v_z$, written as $D_{v_z}$, is defined as $D_{v_z} = E_{v_z} - I_{v_z}$. Let $v_z \in P_1$ and $v_y \in P_2$. The gain $g_{v_zv_y}$ of swapping $v_z$ and $v_y$ is defined as $D_{v_z} + D_{v_y} - 2w(v_z, v_y)$ if $e(v_z, v_y) \in E(G_n)$ and $D_{v_z} + D_{v_y}$ if $e(v_z, v_y) \notin E(G_n)$.

The Kernighan Lin algorithm is iterative in nature. Given a partitioning $P = P_1, P_2$ such that $|P_1| \approx |P_2|$ on the graph $G_n$, all nodes in $P_1$ and $P_2$ are initially unlocked, meaning that they can be exchanged to a different partition. The algorithm then identifies the pair of nodes $(v_z, v_y)$ that have the greatest gain $g_{v_zv_y}$. This pair of vertex is then swapped between partitions and $v_z$ and $v_y$ are locked. The $D$ values of the remaining unlocked nodes are updated. The selection and locking of the pair of nodes with the greatest swapping gain and subsequent updating of the remaining node’s $D$ values is continued until there are no longer any unlocked node pairs remaining. For each pair $(v_{zk}, v_{yk})_k$ of nodes that was selected, the partial sum $S(v_{zk}, v_{yk})_k = \sum_{l=0}^k (g_{v_{zk}v_{yk}})_l$ of the total gain is computed. The pair of nodes such that the partial sum is maximized is identified. All node pair exchanges that occurred after this node pair are undone. This is a single pass of the Kernighan Lin algorithm. Multiple passes are conducted until the maximal partial sum of gains is
zero, meaning that no more improvement on the partition edge cut can be found. A straightforward implementation of this algorithm would have a complexity of $O(n^3)$, where $n = |P_1| + |P_2|$, as it would require $\left(\frac{n}{2}\right)^2$ to find the best node pair to exchange and $\frac{n}{2}$ node pairs exchanged. This leads to a complexity of $O\left(\frac{n}{2}\left(\frac{n}{2}\right)^2\right) = O(n^3)$. An example is given in Fig. 11.2.

Using appropriate data structures and strategies, the complexity of the Kernighan – Lin algorithm can be reduced to $O(n^2 \log(n))$. This is the approach taken in this paper and will be discussed briefly. The nodes assigned to each partition $P_1$ and $P_2$ are inserted into two priority queues $Q_1$ and $Q_2$, respectively, according to their $D$ values, where nodes with greater $D$ values have greater priority within the queue. It is reasonable that the node pair that has the greatest gain value will be near the top of the priority queues $Q_1$ and $Q_2$. Indeed, only a subset of the node pairs need to be explored from $Q_1$ and $Q_2$. Pairs of nodes $(v_{zk}, v_{yk})$ are evaluated in decreasing order of $Dv_{zk} + Dv_{zy}$ and the gain $g_{v_{zk}v_{yk}}$ is computed. The maximum $g_{v_{zk}v_{yk}}$ encountered thus far as the node pairs are being sequentially evaluated is recorded as the current $g_{max}$.

Once a pair of nodes $(v_{zk}, v_{yk})$ is found such that $Dv_{zk} + Dv_{zy} \leq g_{max}$ the search through the nodes pairs is terminated. The node pair $(v_{zk}, v_{yk})$ such that $g_{v_{zk}v_{yk}}$ is the current $g_{max}$, is selected as the node pair to exchange. This approach requires $\frac{n}{2} \log \frac{n}{2}$ to sort the nodes in the priority queues. Since $\frac{n}{2}$ node exchanges are made, the total complexity is $O\left(\frac{n}{2}\left(\frac{n}{2} \log \frac{n}{2}\right)\right) = O(n^2 \log n)$.

The diagonal scanning approach in [200] is used to determine the ordering of node pairs evaluated from $Q_1$ and $Q_2$. An additional strategy is also utilized from [130] to speed up run time. Let $S_{max}$ be the maximal partial sum of gains that has been encountered thus far as the node pairs are being evaluated. If $S_{max}$ does not increase for fifty node exchanges, then the node exchanges are terminated. As before, all node pairs exchanges that occurred after $S_{max}$ are undone.
Figure 11.2: Kernighan-lin. (A) Let nodes $A$ and $B$ be the node pair with the greatest gain. The $E$, $I$, and $D$ values are also shown for $A$ and $B$. (B) The nodes $A$ and $B$ are swapped. The updated edge cut is shown. © 2017 IEEE reprinted with permission
11.2.2.3 Projection of the Partition

After the partitioning \( P = P_1, P_2 \) is found on \( G_n \), it is projected iteratively onto \( G_{n-1} \), \( G_{n-2} \) … \( G_0 \). If \( v_z \) is a node in \( N(G_n) \) assigned to a partition \( P_i \), then its component child nodes in \( N(G_{n-1}) \) will be assigned to \( P_i \) on \( G_{n-1} \). Once a projected partitioning \( P = P_1, P_2 \) is established on \( G_{n-1} \) from \( G_n \), it is refined by the Kernighan Lin algorithm. The partitioning on \( G_{n-1} \) is then projected onto \( G_{n-2} \) and refined.

Recursive bisection is applied to create a partitioning \( P = P_1, P_2 \ldots P_k \) in \( \log_2(k) \) steps, using the greedy graph growing and Kernighan Lin algorithms described earlier. First \( G_n \) is partitioned into two equal partitions that are projected and refined onto \( G_{n-1} \), \( G_{n-2} \) … \( G_0 \). These two partitions are then partitioned into four partitions, which are projected and refined. These partitions continue to be recursively bisected until \( k \) partitions are created. Observe that there is a natural parallelism as the multilevel graph set is being recursively bisected. At each recursive bisection step \( i = 0 \ldots \log_2(k)-1 \), there are \( 2^i \) partitions that can be processed concurrently. The number of partitions that will need to be bisected in a given step \( i \) is \( 2^i \). Thus if \( 2^i \) processors are assigned at each \( i \)th step, the partitioning \( P = P_1, P_2 \ldots P_k \) on \( G_n, G_{n-1} \ldots G_0 \) can be generated in \( \log_2(k) \) steps in contrast to \( k \) number of steps.

11.2.2.4 Global K-Way Kernighan-Lin Refinement Algorithm

After the multilevel recursive bisection with the greedy graph growing and Kernighan Lin algorithm is complete, there is a partitioning \( P = P_1, P_2 \ldots P_k \) on \( G_n, G_{n-1} \ldots G_0 \). This paper follows the global Kernighan Lin heuristic approach in [201] to perform a \( k \)-way refinement on \( P \) for each graph level. Let \( G_i \) be the graph level currently begin refined and \( P = P_1, P_2 \ldots P_k \) be the partitioning on \( G_i \). First, the boundary nodes in \( P \) are identified. A given node \( v_z \) in \( G_i \) is a boundary node in \( P \) if \( E_{v_z} \neq 0 \). The boundary nodes in \( P \) are inserted into a priority queue \( Q \) according to their gains. Here the gain \( g_{v_z} \) of a node \( v_z \) is calculated as \( E_{v_z} - I_{v_z} \). The nodes in \( Q \) are then evaluated in the order of descending gain. Let \( v_z \) be the current node being evaluated and
Let $P_i$ be the partition that $v_z$ is currently assigned to. Let $E_{v_z P_j}$ be the external cost of moving $v_z$ to a partition $P_j$ given by $E_{v_z P_j} = \sum_{(v_z v') \in E(G_i), v_i \in P_i, v \in P_j} w(v_z, v)$. The node $v_z$ is moved to a partition $P_j$ such that $E_{v_z P_j}$ is the maximum external cost out of all of the external costs calculated for each partition. Partition balancing conditions must also be met; a node will not be moved to a partition $P_j$ from a partition $P_i$ if $|P_j| \geq 1.03|P_i|$. As before, let $S_{\text{max}}$ be the maximal partial sum of gains that has been encountered thus far as nodes have been moved between partitions. The k-way refinement algorithm terminates after fifty moves have been made and no improvement to $S_{\text{max}}$ occurs. All node moves that occurred after the $S_{\text{max}}$ was identified are undone. This is a single pass of the global k-way Kernighan Lin heuristic algorithm. Multiple passes are conducted until no more node move improvements can be made. Each graph level can be refined independently by the global k-way Kernighan Lin algorithm with multiple processors.

11.2.3 Graph Partitioning

In this section, distributed graph algorithms for graph simplification, error correction, and graph traversal are introduced.

11.2.3.1 Transitive Edge Reduction

Let $P = \{P_1, P_2, \ldots, P_k\}$ be a partitioning on the hybrid graph $G_0$. Each partition $P_i$ is assigned to a different worker processor. In parallel, each worker processor iterates through the nodes assigned to its partition sequentially. Following the approach described by [4], the edges of each node in $V(G_0)$ are examined to identify transitive edges. Each transitive edge is recorded for removal from the hybrid graph. After each worker processor completes the transitive reduction of its partition, the master process removes the recorded transitive edges from the hybrid graph. If a transitive edge crosses a graph partition, then both of the partitions to which the endpoints of that edge are assigned will record that edge as being transitive. The master process will remove the recorded transitive edge from the hybrid graph.
11.2.3.2 Containment Removal

The graph simplification process removes redundant information in the hybrid graph $G'$. This includes nodes that represent contigs whose sequences are contained within longer contigs represented by nodes in the hybrid graph. To remove these nodes, each worker processor again iterates through the nodes assigned to its partition sequentially. Let $v_y$ be a node that is currently being evaluated. The contig $c_y$ corresponding to node $v_y$ will be aligned to the contigs represented by the neighboring nodes of $v_y$. If $c_y$ is found to be contained within a neighboring contig sequence, then $v_y$ will be recorded for removal from the hybrid graph. This alignment process is also used to detect false-positive edges within the hybrid graph. If the contigs corresponding to the endpoints of an edge in $G'$ have an overlap length less than 50 bps, then that edge will be recorded for removal from $G'$. After each worker process completes the evaluation of its assigned nodes, then the master process removes those nodes from the hybrid graph.

11.2.3.3 Error Removal

Focus employs short dead end path trimming and bubble removal techniques utilized by many assembly tools and described in [103]. Each worker node evaluates its own partition and explores each node to see if it is part of a dead end path or bubble. Nodes that are a part of a short dead end path or bubble are recorded. These nodes are removed from the hybrid graph by the master process.

11.2.3.4 Graph Traversal

Focus constructs contigs by recovering all maximal paths in the hybrid graph. Each worker processor evaluates the nodes in its partition sequentially. Let $v_y$ be a node that is currently being evaluated and is not already in a path. The node $v_z$ becomes the seed for a new path and is first extended by out-edges. If $v_y$ has a single out-edge $e = (v_y, v_z)$ and $e$ is the only in-edge of $v_z$, then $v_z$ is added to the growing path. If $v_z$ is not part of the same partition as $v_y$ or the above criteria is not met, the path extension with out-edges is terminated. The path is then
extended by out-edges from $v_z$ by the same method used for $v_y$. Path extension by out-edges continues until no additional nodes can be added to the path. The path is then extended from $v_y$ by in-edges following the same approach as the extension by out-edges. Once all of the worker processes have finished path extension, the master process joins the sub-paths produced by each worker process. Let $p_1$ and $p_2$ be two sub-paths such that the right endpoint of $p_1$ has an out-edge incident to the left endpoint of $p_2$. If the left endpoint of $p_2$ has no other in-edges incident to it, then the master processor joins $p_1$ and $p_2$. The produced paths are used to construct contigs, which are output by the assembler.

11.3 Results

This results section is organized into five subsections. First the data sets and computational environment are described in detail. In the next section, an experiment is performed to demonstrate the natural parallelism that can be exploited during the graph partitioning process. In section three, it is shown that incorporating biological knowledge improves the partitioning process. Section four describes the speed up curve for trimming and graph traversal algorithms implemented on the distributed hybrid graph. Finally, the results section is concluded by showing that the partitions on the hybrid graph are able to capture features of the metagenomics community structure.

11.3.1 Data Sets and Computational Environment

For the purpose of evaluating the graph partitioning process for the Focus assembler algorithm, three Illumina read data sets were downloaded from the NCBI Sequence Read Archive (SRA) [158]. These Illumina read data sets are sequenced from the gut microbiome of healthy individuals. Details regarding these data sets can be found in table 11.1. All experiments were run on the Crane high performance computing cluster at the Holland computing center [138]. This cluster has 452 nodes with two Intel Xeon E5-2670 2.60GHz processors /16 cores per node. Each node has 64 GB of memory.
Table 11.1: Data Set Characteristics. © 2017 IEEE reprinted with permission

<table>
<thead>
<tr>
<th>Data set</th>
<th>SRA ID</th>
<th>Data set size (GBases)</th>
<th>Read length (bps)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>SRR513170</td>
<td>5.02 Gb</td>
<td>100 bp</td>
</tr>
<tr>
<td>2</td>
<td>SRR513441</td>
<td>4.93 Gb</td>
<td>100 bp</td>
</tr>
<tr>
<td>3</td>
<td>SRR061581</td>
<td>4.97 Gb</td>
<td>100 bp</td>
</tr>
</tbody>
</table>
Prior to graph partitioning, the Focus algorithm was applied to produce both multilevel and hybrid graph sets for each read data set.

11.3.2 Parallel Graph Partitioning

Recall that section IV. C. discussed the natural parallelism of the graph partitioning process. Let $G_0, G_1 \ldots G_n$ be a graph set created by graph coarsening, where $|N(G_n)| \leq |N(G_{n-1})| \leq \ldots \leq |N(G_0)|$. Let $P = P_0, P_1 \ldots P_k$ be a partitioning that is created in $\log(k)$ recursive bisection steps on $G_0, G_2 \ldots G_n$. There are $2^i$ partitions that can be processed concurrently for each recursive bisection step $i = 0 \ldots \log_2(k)-1$. Observe that the maximum number of processors needed to fully take advantage of this inherent parallelism is $2^{\log_2(k)-1}$. Thus for any partitioning of size $k$ for a given graph, the number of processors needed to achieve optimal speedup is $2^{\log_2(k)-1}$.

After recursive bisection of the graph set is complete, the multilevel Kernighan-Lin algorithm further refines each graph in $G_0, G_1 \ldots G_n$. Each graph $G_0, G_1 \ldots G_n$ can be processed independently, thus the number of processors needed to achieve the best speedup overall will be $\max(n, 2^{\log_2(k)-1})$.

Here we present the results of applying an increasing number of processors to create a speedup curve for partitioning the hybrid graph sets of each read data set into 16 partitions. For each number of processors, the graph partitioning algorithm is ran three times. The average of those runs and their standard deviation are shown in Fig. 11.3. Multiple runs were averaged together in this experiment due to time variation introduced by the random seed nodes selected by the greedy graph growing algorithm.

As expected the speedup curve gains begin to level off at about eight to ten processors as $2^{\log_2(16)-1} = 8$ and there are ten graph levels for each of the multilevel and hybrid graph sets.
Figure 11.3: Graph partitioning speedup. Speedup curve for graph partitioning on the three read data sets. © 2017 IEEE reprinted with permission
11.3.3 Incorporation of Domain Specific Knowledge for Graph Partitioning

As stated previously, the multilevel graph set represents full graph uncoarsening to the original overlap graph $G_0$. The hybrid graph set represents a compromise between full graph uncoarsening and incorporation of biological knowledge. Each node in the hybrid graph $G'_0$ represents a cluster of nodes in $G_0$ that are likely to represent consecutive genomic regions within the sample DNA.

In this section, the results from partitioning both the multilevel graph set and hybrid graph set are provided. For each of the read data sets, the multilevel graph sets and hybrid graph sets are partitioned into 8, 16, 32, and 64 partitions in four separate runs. The number of processors used to generate each partitioning is set to $2^{\log(k)-1}$, where $k$ is the number of partitions.

Results from the runs are shown in Fig. 11.4 and Table 11.2. Fig. 11.4 displays the runtimes for partitioning the hybrid and multilevel graph sets for each data set into 8, 16, 32, and 64 partitions. The runtime for partitioning the hybrid graph is shown in blue, while the red bars represent the runtime of the partitioning of the multilevel graph spectrum. Observe that the runtime for partitioning the hybrid graph sets is nearly half of the runtime needed to partition the multilevel graph sets. The edge cut for each partitioning was recorded in Table 11.2. None of the edge cuts were more than 0.43% of the total edge weight of the original overlap graphs.

For each data set and partition size, the edge cut for the hybrid graph set and multilevel graph set is shown. The lowest edge cut for either the hybrid graph set or multilevel graph set is shaded. For all cases except for two, the partitioning of the hybrid graph set produces the lowest edge cut numbers. These results demonstrate the improvement that the partitioning algorithm was able to obtain by the inclusion of biological knowledge. To obtain a partitioning on the original multilevel graph set, the hybrid graph set partitioning can simply be projected onto the multilevel graph set.
Figure 11.4: Hybrid graph set vs. multilevel graph set. Each hybrid and multilevel graph set was partitioned four times with 8, 16, 32, and 64 final partitions. For each graph partitioning, $2^{\log_2(k)-1}$ processors were assigned, where $k$ is the final number of partitions. The runtime is shown for partitioning the hybrid graph set (blue) and multilevel graph set (red) for each read set (1, 2, and 3). © 2017 IEEE reprinted with permission
Table 11.2: Edge Cut for Multilevel and Hybrid Graph Sets. © 2017 IEEE reprinted with permission

<table>
<thead>
<tr>
<th>Part. Num</th>
<th>Data Set</th>
<th>Edge Cut (Hyb.)</th>
<th>Edge Cut (Mult.)</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>1</td>
<td>18878760</td>
<td>17240330</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>72303920</td>
<td>74859760</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>26310110</td>
<td>30489660</td>
</tr>
<tr>
<td>16</td>
<td>1</td>
<td>21097730</td>
<td>21308940</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>80966850</td>
<td>85452040</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>31688290</td>
<td>35577120</td>
</tr>
<tr>
<td>32</td>
<td>1</td>
<td>26994530</td>
<td>25354840</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>84722430</td>
<td>89415260</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>34377760</td>
<td>41504860</td>
</tr>
<tr>
<td>64</td>
<td>1</td>
<td>31898070</td>
<td>32828080</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>90542060</td>
<td>95242730</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>38594120</td>
<td>47502740</td>
</tr>
</tbody>
</table>
11.3.4 Performance of Distributed Graph Algorithms

In this section, the performance of the distributed graph trimming and graph traversal algorithms is described. First the distributed graph-trimming algorithm, which includes transitive reduction, dead-end trimming, bubble popping, and containment removal, was applied to the distributed hybrid graphs for each of the three read data sets. The trimming algorithm was applied for the 8, 16, 32, and 64 partitionings generated for each hybrid graph. Following distributed graph trimming, distributed graph traversal was used to obtain an ordering of the hybrid graphs’ nodes for contig construction. The runtimes for the distributed graph trimming and graph traversal algorithm can be found in Fig. 11.5.

For the distributed trimming algorithm, run time decreased steeply for data sets 2 and 3 as the hybrid graph was split into increasing numbers of partitions. For data 1, the run time also decreased, but not as sharply as data sets 2 and 3. This may be due to the underlying complexity of the input data sets as data set 1 also required much less run time than data set 2 or 3. Graph traversal of all of the hybrid graphs required very little run time, which remained static across the number of partitions of the hybrid graph.

Finally, to assess the consistency of the assembler performance across various partitionings of the hybrid graph, contigs were fully assembled and analyzed from the hybrid graph traversals. The results for each assembly of contigs can be found in Table 11.3.

Table 11.3 demonstrates the consistency of each assembly produced from runs on different partitionings of the hybrid graph. The N50 lengths are very consistent throughout different numbers of partitions. The number of contigs produced for the different partitionings is very similar, only varying by less than a couple hundred contigs for any of the data sets. The maximum length of contigs produced is stable across different partitioning configurations on the hybrid graph. Thus the assembly performance is consistent across different partition configurations of the hybrid graph.
Figure 11.5: Distributed graph algorithms. The trimming and traversal algorithms were applied to the different hybrid graph partitionings. The runtimes for the three read data sets are shown in blue, red, and green. © 2017 IEEE reprinted with permission
Table 11.3: Assembly Statistics. © 2017 IEEE reprinted with permission

<table>
<thead>
<tr>
<th>Data set</th>
<th>Part. Num.</th>
<th>N50 (bp)</th>
<th>Max Contig (bp)</th>
<th>Num. of Contigs</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>4</td>
<td>2 082</td>
<td>25 968</td>
<td>104 185</td>
</tr>
<tr>
<td>1</td>
<td>16</td>
<td>2 083</td>
<td>25 968</td>
<td>104 219</td>
</tr>
<tr>
<td>1</td>
<td>32</td>
<td>2076</td>
<td>25 968</td>
<td>104 470</td>
</tr>
<tr>
<td>1</td>
<td>64</td>
<td>2076</td>
<td>25 968</td>
<td>104 470</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>1 513</td>
<td>10 486</td>
<td>151 411</td>
</tr>
<tr>
<td>2</td>
<td>16</td>
<td>1 514</td>
<td>9 920</td>
<td>151 408</td>
</tr>
<tr>
<td>2</td>
<td>32</td>
<td>1 520</td>
<td>9 920</td>
<td>151 210</td>
</tr>
<tr>
<td>2</td>
<td>64</td>
<td>1 520</td>
<td>9 920</td>
<td>151 210</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>1 286</td>
<td>6 861</td>
<td>117 596</td>
</tr>
<tr>
<td>3</td>
<td>16</td>
<td>1 285</td>
<td>6 861</td>
<td>117 629</td>
</tr>
<tr>
<td>3</td>
<td>32</td>
<td>1 284</td>
<td>6 861</td>
<td>117 632</td>
</tr>
<tr>
<td>3</td>
<td>64</td>
<td>1 284</td>
<td>6 861</td>
<td>117 632</td>
</tr>
</tbody>
</table>
The algorithms implemented to run on the hybrid graph are basic in nature. We plan to expand the number of analysis algorithms that can be applied to the distributed hybrid graph. For example, variant detection algorithms can be implemented to be run on the distributed hybrid graph.

11.3.5 Extracting Biological Knowledge from Graph Partitioning

In this section it is shown that the partitions generated for the hybrid graph built from the three gut metagenomics data sets can capture the community structure of those data sets. First, the gut microbiome reference sequence database for the human microbiome project was downloaded. BWA [71] was used to index this database and align the sequence reads to the reference gut microbiome sequences. The sequence reads were classified to a genus according to their best hits. If no hits were found for a given read, it remained unclassified.

After the alignment of the reads to the gut microbiome database was complete, the major genera for the three data sets were computed. The read classification counts for each of the three data sets were pooled together. The top ten genera that had the greatest pooled read counts were selected for further analysis (Alistipes, Bacteroides, Clostridium, Escherichia, Eubacterium, Faecalibacterium, Prevotella, Parabacteroides, Roseburia). The distribution of these genera across the 16-partitioning for each of the three data sets was analyzed. The distribution of a given genera was calculated as the fraction of its classified reads that correspond to nodes in each partition. The distribution of the major genera can be found in Fig. 11.6 Observe that the distribution of the different genera is not static across the partitions.

Different genera cluster preferentially into different partitions. This is understandable as the nodes representing a contiguous genome are likely to be adjacent to one another in the hybrid graph. Nodes that are highly connected are more likely to be assigned to the same partition. Also, it is notable that many genera that belong to the same phylum often tend to have greater read counts across the same partitions. For example, the genera Roseburia and Clostridium both have
Figure 11.6: Distribution of major genera across partitions. A partitioning of size 16 is represented by the columns of the heat map. Genera are represented by the rows. The colors of the heat map represents the fraction of reads from a genera which is found in a partition. Darker red colors represent a greater fraction of reads. © 2017 IEEE reprinted with permission.
higher fractions of read counts in the same partitions. Both of these genera belong to the phylum Firmicutes. *Eubacterium* is found more frequently in the same partitions as *Rosburia* and *Clostridium* for data sets 1 and 2. The genus *Eubacterium* also belongs to the Phylum Firmicutes. Sequences that are genetically related will have many similar regions of genome that will be represented by interconnected nodes in the hybrid graph. These connected nodes are more likely to be assigned to the same partition.

11.4 Conclusion and Discussion

In this research, the construction of a distributed assembly graph for next generation sequencing data was presented. Unlike most previous approaches, which utilize a distributed de Bruijn graph model, this approach discussed methods for partitioning an extended overlap graph based model.

Three major objectives were covered by this chapter. First a distributed graph model was successfully developed. Multilevel graph partitioning was applied to partition both the multilevel graph set and the hybrid graph set. This partitioning process was applied on three metagenomics data sets. Results demonstrate that the edge cut obtained from the partitioning was never more than 0.43% of the original overlap graph edge weights for both the multilevel graph set and hybrid graph set. Several distributed graph algorithms were implemented on the distributed hybrid graph. Results demonstrated a substantial speedup for the graph trimming algorithm. The graph traversal algorithm did not show a great speedup; however, this algorithm had a very fast runtime. Finally, assembly results obtained from different graph partitionings were consistent, demonstrating that assembly quality is not affected by partitioning the hybrid graph.

The second objective was to demonstrate that the integration of prior biological knowledge into a naïve graph partitioning algorithm could improve its performance. Partitioning the multilevel graph set represents the original naive partitioning algorithm since the multilevel graph set is fully uncoarsened to the overlap graph during partitioning. Partitioning the hybrid
graph set and projecting that partitioning onto the original multilevel graph set represents a compromise between full uncoarsening and incorporation of biological knowledge. The runtime for partitioning the hybrid graph set was roughly half of the runtime for partitioning the full multilevel graph set. Finally, the edge cut was improved in most cases for the hybrid graph set in comparison to the multilevel graph set.

The third objective was to demonstrate that biological knowledge could be obtained from the graph partitioning. It was shown that the distribution of genera was not equal across partitions, but that nodes representing reads from a given genus tended to be assigned to the same partition. Furthermore, phylogenetically related genera were also often found in the same partition.

This research demonstrates high performance computing techniques for information extraction from big data. Typically, high performance computing focuses solely on faster runtimes. High performance computing is also a tool for extracting usable knowledge from big data that would have other been impossible with limited computing resources. Finally, this research demonstrates the importance of taking input data characteristics into consideration when designing and applying algorithms. It was shown that integrating biological knowledge into the naïve partitioning process can greatly improve its results. We anticipate that this prior knowledge integration approach can improve numerous other naïve computational algorithms.
CHAPTER 12

PARALLEL FOCUS ASSEMBLY: ENERGY-AWARE SHORT READ ALIGNMENT UTILIZING HIGH PERFORMANCE COMPUTING

12.1 Introduction

This chapter explores the energy aware approaches for assembling next generation sequencing. As the wealth of biological data and associated computational burden continues to increase, there has become a need for the development of energy efficient computational approaches in the bioinformatics domain. To address this issue, an energy-aware scheduling (EAS) model has been developed [202] to run computationally intensive applications that takes both deadline requirements and energy factors into consideration. An example of a computationally demanding process that would benefit from the scheduling model is the assembly of short sequencing reads produced by next generation sequencing technologies. Next generation sequencing produces a very large number of short DNA reads from a biological sample. Multiple overlapping fragments must be aligned and merged into long stretches of contiguous sequence before any useful information can be gathered. The assembly problem is extremely difficult due to the complex nature of underlying genome structure and inherent biological error present in current sequencing technologies. We apply the EAS model to the read overlapper of the Focus algorithm, giving us the ability to generate speed up profiles. The EAS model was also able to dynamically adjust the number of nodes needed to meet given deadlines for different sets of reads. We conduct multiple experiments to evaluate the computational resources needed to complete the overlapping process while balancing task deadline requirements with energy minimization. These experiments demonstrate the viability of the proposed energy-aware scheduling model and characterize the impact of various parameters on program runtime.

12.2 Methods

12.2.1 Energy Aware Scheduling

Scheduling is a classical field with several interesting problems and results. Due to its wide range of applications, the scheduling problem has been attracting many researchers from a number of different fields. A scheduling problem emerges whenever there is a choice. This choice could be the order in which a number of tasks can be performed and/or in the assignment of those tasks for processing. In general, the scheduling problem assumes a set of resources and a set of consumers serviced by these resources according to a certain policy. Given a set of customers, resources, and constraints, a solution to the scheduling problem attempts to find an efficient policy (schedule) for customer access to resources while optimizing some desired performance measure such as the total service time (schedule length).

Over the years several methods have been used to address the sequencing problem including complete enumeration, heuristic rules, integer programming, and sampling methods. It is clear that complete enumeration is impractical because the problem is exponential; hence optimal solutions cannot be obtained in real time [203][204]. However, many heuristic methods have been successfully applied to most general cases of the scheduling problem. Such methods include traditional priority-based algorithms [205], task merging techniques [206], critical path heuristics [205][207]. In addition, distributed algorithms have been designed to address different versions of the scheduling problem [208].

Energy aware scheduling is a special case of the general scheduling problem in which our scheduling policy is the optimization of energy in HPC systems or battery power in mobile devices. Minimizing the power utilization which is directly proportional to costs becomes the most important consideration in a system that is energy aware. At the same time this system must still meet other specified parameters such as task deadlines.
Figure 12.1: Energy Aware Scheduling System. © 2012 IEEE reprinted with permission
Simply put, an energy aware scheduling system is a scheduling problem that assumes a set of resources and a set of consumers serviced by these resources according to an energy aware policy. Given a set of customers, resources, and constraints, a solution to the energy aware scheduling problem attempts to find an efficient policy for customer access to resources while optimizing battery power utilization. Accordingly, an energy aware scheduling system can be considered to consist of a set of consumers, a set of resources, and an energy aware scheduling policy as shown in figure one. Clearly, there is a fundamental similarity to scheduling problems regardless of the difference in the nature of the tasks and the environment.

12.2.2 Read Overlap Detection

For the Focus assembler, we have developed different modules for the read overlapper. The default approach of the read overlapper is to determine all overlaps between reads. For high coverage data sets, an alternate module has been developed that first clusters and removes reads that have containment overlaps with other reads in the data set. The EAS model was applied to this high coverage read overlapper module. In this section, we provide a description of the three-step approach for read overlap detection for high coverage data sets. The first step orders a read dataset $S$ in descending read length and partitions it into subsets. The second step maps each read that forms a containment overlap with one or more other reads to a suitable representative read following a hierarchical clustering scheme introduced by CD-Hit [209]. After clustering is complete, the final step identifies dovetail overlap relationships among the remaining representative reads.

12.2.2.1 Read Preprocessing

The containment clustering step of the overlap detection phase requires that the reads are sorted by descending length. First the reverse complements of an input read dataset $R$ are generated to form the read set $S = (R, \bar{R})$. It then sorts $S$ into descending order of length by a merge sort algorithm, and partitions $S$ into $n$ subsets = $\{S_0, S_1, \ldots S_{n-1}\}$ of size $m$, where $n$ is
specified by the user. Each read subset $S_k$ is sorted in descending read length and the subsets are ordered such that $\text{readLengths}(S_0) \geq \text{readLengths}(S_1) \geq \ldots \geq \text{readLengths}(S_{n-1})$. Fig. 12.2 displays a flow chart model of the read preprocessing step.

12.2.2.2 **Containment Clustering**

The initial read clustering step follows the greedy hierarchical clustering scheme introduced by the CD-hit algorithm [209]. The longest read becomes the first representative. It is used to search for containment overlaps among the remaining reads using the exact matching and alignment methods described in the section three. If a read forms a containment overlap with the current representative and its alignment meets minimum length and alignment identity requirements, it is mapped to that representative read. The algorithm considers each read in the order of descending length. If a read is not already mapped to an existing representative, it becomes a new representative read and is used to query the remaining reads in the dataset for containment overlaps. A read that has been mapped to a previous representative read but forms a containment overlap with the current representative is remapped to the current representative if its alignment identity with the current representative is greater than its alignment identity with the previous representative. After this process has completed, all read to representative mappings are recorded for use in the consensus phase of the assembly process. Fig. 12.3 provides an example of the containment clustering process.

12.2.2.3 **Dovetail Overlaps**

After containment clustering is complete, the remaining representative reads are used to query the read dataset for dovetail overlaps with other representative reads. The exact matching and alignment methods of section three are used to locate dovetail overlap relationships. If a dovetail overlap meets minimum alignment length and alignment identity requirements, it is recorded for use in the graph construction phase of the assembly algorithm.
Figure 12.2: Pre-processing step. © 2012 IEEE reprinted with permission
Figure 12.3: Containment clustering. Reads two and four cluster to read one. Read five clusters to read three. © 2012 IEEE reprinted with permission
12.2.3 Parallel Implementation and EAS Model

The input read dataset $S$ is partitioned into $n$ subsets $= \{S_0, S_1, \ldots S_{n-1}\}$ of size $m$ during the initial read sorting and preprocessing step. A master thread sends each unique subset combination of size two as input to worker processors running serial versions of the containment clustering and dovetail overlapping algorithms. The master thread manages the execution order constraints of the containment clustering step.

12.2.3.1 Solution Overview

The EAS engine runs the pre-processor on the input fasta file, the output of which is the $n$-split read subsets. Let us assume that the large file has $m$ sequences, then each of the smaller files will contain $(m/n)$ sequences in sorted order. This is shown in Fig. 12.4.

The files created in the pre-processing step become inputs to the EAS engine. The EAS engine runs the alignment program in a 2-step process. The first step finds the containment overlaps and the second step determines the dovetails overlaps among the remaining representative reads. The containment part of the execution is not naively parallel; the execution of certain pairs of subsets (tasks) has to be done in order, only then can dependent subsets be processed. The main process flow is shown in Fig. 12.4.

12.2.3.2 Containment Execution – Step 1

The execution dependencies are shown in Fig. 12.5 for the following set of containment tasks $T = \{(0, 0), (0, 1), (0, 2), (0, 4), (1, 1), (1, 2), (1, 3), (1, 4), (2, 2), (2, 3), (2, 4), (3, 3), (3, 4), (4, 4)\}$, where each integer represents a read subset. The tasks along the diagonal $(0, 0), (1, 1) (2, 2), (3, 3)$ and $(4, 4)$ are considered to be higher priority tasks because they have a greater number of child/dependent tasks. All other tasks have a normal priority in terms of execution. After a task gets released, meaning that all of its predecessors have been executed, it is sent to the EAS execution queue. When the task has completed executing, the EAS engine checks to see if any dependent tasks can be released for execution.
Figure 12.4: Process flow diagram. © 2012 IEEE reprinted with permission
Figure 12.5: Execution dependencies of containment tasks. © 2012 IEEE reprinted with permission
Now let us take a look at the example in Fig. 12.5 where we have five read subsets. When the task (0, 0) is complete, it releases all the tasks in that row which are tasks (0, 1), (0, 2), (0, 3) and (0, 4). It cannot release (1, 1) because task (1, 1) still has another dependency on (0, 1). When (0, 1) is completed, it will release task (1, 1). Completion of task (1, 1) will flag (1, 2), (1, 3), and (1, 4) but they will only be released when both (1, 1) and the tasks above them namely (0, 2), (0, 3), and (0, 4) have completed execution. This will continue until all tasks are executed. The last task to be executed will be task (4, 4) in our example. Note that the total number of tasks executed would be fifteen. This can be calculated easily using equation one. We would like to point out that the containment phase is bounded by the number of files (in this case five). We cannot use more than five nodes at any given time due to task dependencies even though we have a total of fifteen containment tasks.

12.2.3.3 Dovetail Execution – Step 1

The execution dependencies of the dovetail tasks are much more straightforward than those for the containment tasks. The dovetail tasks do not have any dependencies on each other and hence can be run in a naively parallel way, allowing us to use as many processors as possible. Continuing with our previous example with fifteen tasks, we could execute (0, 0), (0, 1), (0, 2), (0, 3), (0, 4), (1, 1), (1, 2), (1, 3), (1, 4), (2, 2), (2, 3), (2, 4), (3, 3), (3, 4), (4, 4) all at the same time during the dovetail phase.

The total number of tasks that need to be executed in each of the above steps (containment and dovetail steps) is given by the equation below, where \( n \) is the number of read subsets and \( T \) is the total number of tasks.

\[
T = \frac{n(n + 1)}{2}
\]

(1)

12.3 Results

We downloaded *Escherichia coli* \( W \) reads produced by the 454 Titanium technology from the NCBI [210] sequence read archive (accession no. SRR060736 and SRR060737, made public
by JCVI). The sequences were trimmed to remove adaptors. The final result was 337,294 trimmed reads. For our experiment in the pre-processing step we decided to split these into 16,866 sequence reads per file, i.e. read subset (except for the last file which contained 16,814 reads). This resulted in 40 files and a total of 674,588 reads. (The preprocessing step generates the reverse complement of each read.) We then used the EAS engine to run the assembly algorithm using 1 to 31 nodes. For our experiments we used the HPC environments available at UNO (University of Nebraska at Omaha). We initially start out with the Blackforest cluster (16 nodes) and then move to a true commercial strength HPC named Firefly cluster (1100 nodes) at the Holland Computing Center [138].

**Firefly Cluster:** The firefly cluster is a large commercial strength cluster at the Holland Computing Center which comprises of 1,151-node supercomputer cluster of Dell SC1435 servers. Each node contains two sockets, and each socket holds a quad-core (four 64-bit AMD Opteron 2.2 GHz processors). The computational network utilizes an 800 MB/sec Infiniband interconnect. Each node has its own 8 GB of memory, and 73 GB of disk space.

Chart (a) in Fig. 12.6 shows the execution time of the algorithm in seconds versus the number of nodes used for each run. It shows that after 11 to 12 nodes we do not see any significant performance gain. Along with the total execution time, we captured the average execution time per worker node and the overhead. We find that as we increase the number of nodes the overhead curve follows the execution time curve. It is important to note that in a HPC a significant portion of the master process’s work is distributing the tasks and managing the task dependency among the worker processes along with handling of the communication between master and worker processes. This is clearly depicted by chart (b) in Fig. 12.6. It is important to note that given the nature of the task dependencies in the containment phase not all nodes are working all the time, and hence we see a smaller overall curve for the average worker time per node. This leads us to ask the question, “How parallelizable is the program?”
Figure 12.6: Chart (a): EAS - Execution time v/s Nodes. Chart (b): Execution time/Overhead v/s nodes. Chart (c): Speedup curve for the assembly program © 2012 IEEE reprinted with permission
For the purpose of answering this question we plotted the program speedup against the number of nodes and integrated this curve with a plot of Amdahl’s law in chart (c) in Fig 12.6. Amdahl's law is defined by the formula:

\[
\frac{1}{(1 - P) + \frac{P}{N}}
\]

As \( N \) approaches infinity, the maximum speedup tends to \( 1/(1 - P) \). In practice, performance/price falls rapidly as \( N \) is increased once there is even a small component of \( (1 - P) \). A great part of the craft of parallel programming consists of attempting to reduce \( (1 - P) \) to the smallest possible value.

We can conclude that the overlap detection algorithm of the Focus assembler has a speedup between 20 - 25 times (which is between 90% - 95% parallelizable).

Next we set up experiments to see if the EAS engine would be able to dynamically adjust the number of nodes to meet a given deadline. We used four groups of read datasets generated from SRR060736 and SRR060737. Each group was partitioned into a different number of files as shown in table 12.1.

Each group of files was ran against five different deadlines (30, 60, 90, 120, and 150 minutes). Each of these jobs was assigned a starting number of nodes by the EAS engine based on the run profile/speedup curve. As the tasks were completed, variances between EET (Expected Execution Time) and AET (Actual Execution Time) resulted in the EAS engine adjusting the number of nodes up (+N) or down (−N), if there were equal number of (+N) and (−N) adjustments it resulted in a net (0) adjustment and finally the scenario of no adjustments being made (−). The experimental results showed that the EAS engine was able to dynamically adjust nodes to minimize energy utilized while meeting the deadlines. Figure 12.7 displays the node adjustment results.
<table>
<thead>
<tr>
<th>Group</th>
<th>Number of Files</th>
<th>Number of Sequences</th>
</tr>
</thead>
<tbody>
<tr>
<td>G1</td>
<td>5</td>
<td>84330</td>
</tr>
<tr>
<td>G2</td>
<td>10</td>
<td>168660</td>
</tr>
<tr>
<td>G3</td>
<td>15</td>
<td>337320</td>
</tr>
<tr>
<td>G4</td>
<td>20</td>
<td>674588</td>
</tr>
</tbody>
</table>
Figure 12.7: EAS engine - dynamic node adjustments. © 2012 IEEE reprinted with permission
12.4 Conclusion and Discussion

Based on the results we can clearly observe that given a deadline we can choose the appropriate number of nodes to run the overlap detection phase of the assembler on based on our new understanding of the run-profile we just produced. This will allow us to apportion just enough nodes to meet the deadline thus maximizing the objective of performance with minimum energy utilization. We also observed that with a smaller number of nodes we have larger gains in performance and above a certain number of nodes the performance gain is only modest at best. In fact as we add additional nodes our communication costs and related overhead is higher.

Clearly different bioinformatics applications and algorithms will have different run profiles and understanding each one of them will allow us to best assign the appropriate number of nodes to meet a given deadline. It was also important to see how the number of read subsets impacted the performance/energy criterion. Our experiments suggest a bowl shaped curve when we varied the number of files for the same number of nodes. There must be some optimum value for the number of files for each set.

This research highlights the importance of understanding the degree of parallelism for the program, which is done by establishing the run profile/speedup curve. The EAS engine uses the knowledge from the run profile to make intelligent and dynamic decisions about number of nodes to use to minimize energy utilization and still provide necessary performance. Clearly it is no longer sufficient to simply run a program in a HPC environment. It is important and essential to understand the data, its characteristics, and the application domain to build a parallel program that is energy aware.
CHAPTER 13
DISCUSSION AND FURTHER DIRECTIONS

13.1 Conclusion

In conclusion, this dissertation presents a novel, graph theoretic model called Focus for the assembly and analysis of next generation sequencing reads. This novel graph theoretic assembler differs from previous approaches in numerous ways. The most important of these differences is the ability of the Focus assembler to utilize the assembly graph to extract biological features in an analysis-based approach. This research also focused on several other important innovations including customization of the assembly process and the development of intelligent high performance computing methods.

The Focus assembler presents a new graph model that has not been implemented in previous assemblers. In contrast to utilizing a single assembly graph to model the reads and overlap relationships, the Focus assembler uses a set of graphs. This graph set models the reads and their overlap relationships at various levels of information, capturing both local and global features in the read data set. This multigraph set affords the Focus assembler several advantages. Here we revisit Fig. 1.1 in Fig. 13.1 illustrating the advantages of the Focus assembler versus previous approaches.

In contrast to previous assemblers, the multilevel graph affords the Focus assembler the ability to capture both local overlap relationships between reads as well as global relationships between groups of reads. These global relationships reveal biologically significant features such as repetitive and conserved elements. The multiple graph set provides multiple views of these global and local features in contrast to a single graph, which provides only a local view of individual read overlap relationships. This ability to capture information across a spectrum of granularity gives the Focus assembler flexibility in contrast to the static models of its predecessors. Finally, the large data sets produced by next generation sequencing technologies
Figure 13.1: Innovations of the Focus assembler in comparison to previous assembly approaches. Fig. 1.1 revisited.
necessitates the use of high performance computing by most assembly tools. These assembly tools typically apply naïve, data-independent approaches for high performance computing. Data-independent approaches are unlikely to perform optimally for all data sets. Further, in most applications high performance computing is applied purely to obtain results in a shorter time frame. However, this research demonstrated that high performance computing could reveal insights into the data on which it was applied.

To formalize the advantages of the Focus assembler in contrast to previous approaches, three problem statements were defined and characterized in this research. Several specific aims were developed to address each of the problem statements above. This dissertation presented the results of addressing each of the specific aims developed during the course of this research. These problem statements and associated specific aims are briefly reviewed below.

Problem Statements

- **Problem 1**: Pure assembly versus data analysis and feature extraction
- **Problem 2**: Data independent assembly of next generation sequencing reads
- **Problem 3**: Massive data set sizes and naïve parallel assembly approaches

Specific Aims

- **Aim 1**: Develop an innovative graph-theoretic algorithmic platform
- **Aim 2**: Apply graph data-mining to extract biologically relevant features
- **Aim 3**: Customize the assembly process to produce improved assembly results
• **Aim 4:** *Develop intelligent high performance computing approaches for NGS assembly*

First before problem statements 1-3 could be addressed, the Focus assembly pipeline had to be developed. This was accomplished through specific aim 1. The Focus software tool was developed as a complete assembler pipeline. The Focus assembler is composed of several modules. First, the Focus assembler contains an in-house preprocessor that trims reads according to user-input trimming lengths as well as by read quality value. This preprocessor also formats and subsets the input read data set for alignment by the parallel read alignment module.

The read alignment module accepts the trimmed reads from the preprocessor and uses suffix arrays with a k-mer search and extension approach to detect overlap relationships between reads. The read alignment module is capable of running in parallel in high performance computing environments to scale to researcher needs.

Once the read-overlapping module is completed, the overlaps are loaded into an overlap graph for the multilevel graph construction module of the assembler. Heavy edge matching is used to create the multilevel graph set that models the read data set across a spectrum of granularity.

The hybrid graph set module integrates the information from different graph levels to construct the hybrid graph. Once the hybrid graph is constructed it is trimmed and error corrected. This hybrid graph serves as the foundation for short read analysis, feature extraction, and assembly.

Once the Focus assembler was developed, aims 2-4 were addressed. In aim 2, it was demonstrated that the Focus assembler is a strong platform for data analysis. Biologically relevant features such as mobile elements were shown to be captured in the structure of the hybrid graph.
Graph-mining was successfully applied to identify transposase–associated gene features in metagenomics data sets obtained from the gut microbiomes of healthy individuals and patients with Crohn’s disease. For aim 3, it was demonstrated that the Focus assembler could be customized according to input data set characteristics. By recasting the graph theoretical foundation of the Focus assembler from the interval graph to the tolerance graph, we were able to adjust individual node parameters according to local characteristics such as node degree and read length. By dynamically adjusting to the input data set characteristics, the Focus assembler was able to produce improved results. Finally to address the growing throughput of next generation sequencing technologies, intelligent and domain-specific high performance computing approaches for assembling next generation sequencing reads were developed in aim 4. Unlike previous approaches that are naïve in nature, the Focus algorithm integrates properties of the input data to improve computational time and results. By integrating information into graph partitioning methods, we are also rewarded as the graph partitioning was shown to provide insights into metagenomics community structure. An energy-aware approach was applied to the read overlapping portion of the Focus assembler. The ability to balance energy consumption with deadlines is important, especially as sequencing costs continue to fall and data production continues to increase. Previous assemblers do not take energy-awareness into consideration, leading to unnecessary expenditure of computational resources.

13.2 Discussion

In its earlier stages bioinformatics was mostly computationally focused, with little integration of biological knowledge. Computational models and methods were developed in a theoretical fashion with biological problems serving as case studies rather than drivers of algorithm development. Bioinformatics tools are often seen as a black box, where input data undergoes a series of transformations that results in the production of output data that may or may not be correct. This black box approach is inflexible and cannot dynamically adapt to input data set characteristics. Researchers using these bioinformatics tools as black boxes with inadequate
understanding of their internal workings and performance characteristics may obtain results that are incorrect, leading to incorrect conclusions in downstream analysis.

The assembly of next generation sequencing reads can be seen as a black box operation by many researchers. Assembler tools accept input data and then perform a series of operations to produce output data that may or may not be correct. During the assembly process information is lost as reads are merged into contigs. The individual and global overlap relationships between read and groups of reads are not represented in the flat contig fasta files that are produced as output. In order to construct contigs, assemblers may have to choose between alternate possible assemblies. These choices made by the assembler are not visible to the researcher applying the algorithm.

The Focus assembler was developed to mitigate this concept of black box assembly. Rather than assembling to completion and producing contig flat files, the Focus assembler allows for mining of features directly from the hybrid graph, minimizing information loss. The assembly graph contains all information that is relevant to the read data set. Regions difficult to assemble such as repetitive regions (ex. mobile elements) are typically ignored by previous assembly approaches or are output as highly fragmented contigs in the fasta flat file with little to no global context of their orientation. The multilevel graph set of the Focus assembler facilitates the ability to select the desired level of assembly. Each increasing level of the multilayer graph set represents a more complete assembly. However, as an increasing number of nodes are merged, more error is introduced into the assembly graph. Thus it would be beneficial to be able to select a partial level of assembly completeness. This is discussed further in the future directions of the Focus assembler. In this research, the concept of algorithm customization is also explored. The ability to customize bioinformatics algorithms according to input data set characteristics will move them from black box approaches to intelligent and dynamic solutions that adapt to researcher needs.
13.2 Limitations

While this research was expansive, next generation sequencing data is extremely information rich. It would be nearly impossible to cover all possibilities for next generation sequencing applications in a single dissertation. Thus several limitations of this research exist.

First, the size of the data sets is limiting in these studies. For example, in the graph-mining section there were eight metagenomics data sets from healthy individuals and five metagenomics data sets from Crohn’s disease patients. It would be beneficial to reproduce the graph data-mining study with hundreds of metagenomics data sets from individuals with Crohn’s disease and healthy controls. This would provide much better statistical power to the study.

The graph-mining experiments for determining antibiotic resistance genes spread in metagenomics data from Crohn’s and health gut microbiomes would also benefit from experimental validation. PCR analysis should be applied to validate the presence or absence of certain antibiotic resistance genes in the Crohn’s disease or healthy samples.

The application of this assembler has primarily been conducted on public data sets. It has been limited to a testing and feasibility stage. This assembler lacks application with current clinical data sets. The Focus assembler tool needs to be expanded for use by outside researchers.

As we continue to expand and develop our graph-mining platform, it will be limited by the completeness of reference databases. If a genome has never been sequenced and contains a novel biological feature, it would be impossible to know which graph structure would correspond to the novel biological feature. Thus far, this study has been limited mostly to studying graph structure associated with antibiotic resistance genes. This work should be expanded to take other biological features into account for graph mining.

Finally, there are limitations with the assembly of next generation sequencing reads in general due to the complexity of the data. As read technologies continue to grow in length it will be feasible to span repetitive regions not previously amendable to sequencing with short read
technologies. Thus the quality and number of completed genomes will continue to increase as reads lengthen and sequencing technologies advance.

### 13.3 Future Directions

There are many different directions in which the Focus assembler can evolve. Here a subset of more immediate future directions is explored. Following from the discussion above, the concept of partial assembly is particularly amenable to the Focus assembler. For many applications, it might not be required to assemble the entire genome. As an example, to extract the gene content of a metagenomics sample, reads would only have to be assembled into contigs as long as the longest genes. By only partially assembling data, one may avoid mis-assembly while obtaining all of the required information needed. The structure of the multilevel graph sets provides many natural cutoff levels for the assembly process. One could assemble from the more granular graphs in the set, resulting in more accurate but more fragmented contigs. Assembling from later graphs in the multilevel graph set would produce a more complete assembly but at the cost of possibly introducing more errors. In the next steps of the Focus assembler, it is planned to evaluate partial assembly and feature extraction. In particular, we plan to optimize the level of assembly that allows for optimal feature extraction while minimizing contig error rates.

The second path that we will pursue is the concept of utilizing the assembly graph for metagenomics classification and target read enrichment with an example in Fig. 13.2 Currently, there are many unknown environmental microorganisms. Many applications require extracting whole genomes from metagenomics data sets. However, due to the lack of complete metagenomics reference databases, not all reads in a target genome will map to the reference database in many cases. To address this issue, we will use the assembly graph for target read enrichment. This will be accomplished by mapping the reads in an input data set to a metagenomics reference database. These reads will also be loaded into an assembly hybrid graph by the Focus assembler. Each node will be labeled with the reference genome, if any, that its
Figure 13.2: Assembly graph target read enrichment. A) A labeled seed node whose read maps to a reference genome is shown in red. Other nodes without labels are shown in blue. B) The seed node influences the labeling of its surrounding unlabeled nodes. Nodes that are closer to the seed node in the assembly graph are assigned a more confident label. Nodes that are a further distance in the assembly graph receive a less confident label.
corresponding read mapped to. Once all of the nodes are labeled, each of the labeled nodes can “influence” its surrounding nodes to join its labeling. The more labeled nodes belonging to a given reference genome within a close distance to an unlabeled node, the more likely that the unlabeled node will also belong to that same labeling. Please see Fig. 13.2 for an illustration on how labeled nodes will influence their unlabeled neighbors.

Additional future directions include the modification of the Focus assembler to accept RNA-seq data. This will require the Focus assembler to be able to assemble alternate gene isoforms and identify splice junctions. In this implementation, the Focus assembler may also be modified to detect fusion genes. This will have important implications for cancer research. The Focus assembly tool is an intelligent, dynamic, and customizable program that will continue to adapt and evolve to address researcher needs.

To facilitate the use of this assembler by the public, work will be conducted to increase the usability of the Focus assembler. This includes writing detailed manuals describing how to run Focus. Improvements to command line helps screens will also be made. Beyond this, future steps for usability might include the development of a web interface to access Focus and/or the release of Focus on GitHub. By releasing the Focus tool publicly and improving usability, Focus has the potential to make a major impact in numerous research projects as it was developed to be flexible and adaptable to researcher needs.
REFERENCES


[17] 


