Optimizations of De Novo Genome And Metagenome Assembly

Author: Christopher Bun

Adviser: Dr. Rick Stevens

July 26, 2013
Abstract

Since the advent of genetic sequencing technologies, the genome assembly problem has become a widely studied topic, for the information gained from an organism’s completed genome offers myriad biological insights. Most early genome projects that were completed, beginning with the first microbial genome in 1995 through the Human Genome Project in 2001, were high in cost and low in throughput. Recently, technologies have led to the so-called era of “Next-generation sequencing,” in which sequencing cost is plummeting at a rate much faster than computation costs, creating a genomic data crisis. Furthermore, cost and throughput benefits are gained at the penalty of data usefulness, introducing obstacles into the genome assembly computational problem.

Accurate genome assembly is essential to many biologic studies but its execution is far from trivial. Base calling errors, indels, biases, translocation, and repeat regions are only a portion of the challenges to be overcome. Many tools and methods have been devised, but completely automatic and accurate protocols are still imperfect; thus a solution to this deluge of new data remains at large.

This thesis introduces AssemblyRAST, a service-oriented framework based on a variety of open-source technologies that allows for workflow flexibility, extensibility, and automation, for rapid genome and metagenome assembly, analysis, and method discovery. By utilizing many of the known processing algorithms, we attempt to study and achieve optimal assembly of next-generation sequencing data through dynamic and integrative pipelines.
## Contents

1 Introduction  

2 Background  

2.1 Sequencing Technologies  

2.1.1 Illumina  

2.1.2 Roche 454  

2.1.3 Life/APG SOLid  

2.1.4 Supportive Sequencing Information  

2.1.4.1 Quality/Confidence Scoring  

2.1.4.2 Pair Sequencing  

2.2 Genome Assembly  

2.2.1 Challenges  

2.2.1.1 Next Generation Sequencing  

2.2.1.2 Sequencing Error  

2.2.1.3 Genetic Properties  

2.2.1.4 Metagenomes  

2.2.2 Assembly  

2.2.2.1 Overlap-Based Assembly  

2.2.2.2 Greedy Extension  

2.2.2.3 Overlap Layout Consensus (OLC)  

2.2.2.4 De Bruijn Graph  

2.2.3 Preprocessing  

2.2.3.1 Error Correction  

2.2.3.2 Artifact Removal  

2.2.4 Postprocessing  

2.2.4.1 Scaffolding  

2.2.4.2 Gap Closing  

2.2.5 Meta-Algorithms  

2.2.5.1 A5  

2.2.6 Assessment  

2.3 Computing Systems  

2.3.1 Scientific Compute Services  

2.3.1.1 RAST  

2.3.1.2 MG-RAST  

2.3.1.3 KEGG
### 3 Implementation

#### 3.1 Introduction

#### 3.2 Assembly Pipelines

- **3.2.1 Unified Assembly Invocation**
- **3.2.2 Explicit Pipeline Invocation**
- **3.2.3 Automatic Pipeline Branching**
- **3.2.4 Automatic and Dynamic Optimization**
- **3.2.5 File Handling**

#### 3.3 Visualization and Rapid Analysis

#### 3.4 System Design and Infrastructure

- **3.4.1 System Overview**
- **3.4.2 Control Server**
  - **3.4.2.1 Request Interface**
  - **3.4.2.2 Job Delegation**
  - **3.4.2.3 User Space**
  - **3.4.2.4 System and Cloud Monitoring**
- **3.4.3 Compute Runtime**
  - **3.4.3.1 Queue Listening**
  - **3.4.3.2 Plugin Framework**
  - **3.4.3.3 Data and Efficiency**
- **3.4.4 Workload, Robustness, and Scalability**
- **3.4.5 Data**

#### 3.5 User capability

- **3.5.1 RESTful API**

### 4 Results and Analysis

#### 4.1 Example Assemblies

- **4.1.1 Multiple Assembler Comparison**
  - **4.1.1.1 Data**
  - **4.1.1.2 Evaluation Metrics**
  - **4.1.1.3 Assemblers**
  - **4.1.1.4 Comparisons**
  - **4.1.1.5 Commands**
  - **4.1.1.6 Discussion**
- **4.1.2 Read and Assembly Processing Stages**
  - **4.1.2.1 Discussion**
- **4.1.3 Parameter Search**
- **4.1.4 Hybrid Assembly Merging**
  - **4.1.4.1 Discussion**

### 5 Discussion

- **5.1 Future**
List of Figures

2.1 3-mer De Bruijn Graph .............................................. 16
2.2 4-mer De Bruijn Graph .............................................. 17
2.3 NGA\texttimes plot of V. cholera assembly by Velvet parameter sweep ........................................................................ 18
3.1 Assembly and Analysis Workflow ........................................ 27
3.2 The Assembly RAST Infrastructure ....................................... 32
4.1 Visual comparison of GC content for V. cholerae HiSeq Assemblies ................................................................. 39
4.2 NGA\texttimes score plot for V. cholerae HiSeq Assemblies ................................................................................. 40
4.3 Runtimes of assemblers for various data sets ................................. 41
4.4 Relative ALE Scores of V. cholera assembly .................................... 45
4.5 NGA\texttimes plot of V. cholera assembly ........................................ 46
4.6 NGA\texttimes plot of V. cholera assembly by Velvet parameter sweep ........................................................................... 52
4.7 Velvet Hash Length vs. NGA50 Score on Rsp HiSeq and MiSeq ........................................................................ 53
4.8 NGA50 scores of pairwise mergings of V. Cholerae assemblies using GAM-NGS. The principle diagonal represents assemblies before merger. ........................................ 54
List of Tables

2.1 Profile of Major Next Generation Sequencing Technologies ............. 8
2.2 FastA Base Pair Codes .................................................................................. 11

4.1 Comparison of NG A50 scores. Best values are shown in bold. NG A50 score is described in section ( *A6 was used. ................................................. 44
4.2 Comparison of misassemblies ........................................................................ 44
4.3 All statistics of R. spaeroides assemblies .............................................................. 47
4.4 Effects of preprocessing on V. cholera NG A50. The best scores per assem- bler are shown in bold. Fields with '-' indicate an error generated by the assembler ........................................................................................................ 51
4.5 Effects of preprocessing on V. cholera misassembly. The fewest misassem- blies per assembler are shown in bold ................................................................. 51
4.6 Data statistics for Velvet parameter sweeps. *N50 score ....................................... 52
## Listings

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1 Velvet Command</td>
<td>27</td>
</tr>
<tr>
<td>3.2 Kiki Command</td>
<td>27</td>
</tr>
<tr>
<td>3.3 IDBA Command</td>
<td>27</td>
</tr>
<tr>
<td>3.4 Example Invocation of Assemblers</td>
<td>28</td>
</tr>
<tr>
<td>3.5 Example Pipeline</td>
<td>28</td>
</tr>
<tr>
<td>3.6 Example Pipeline with Explicit Parameters</td>
<td>28</td>
</tr>
<tr>
<td>3.7 Pipeline Branching</td>
<td>29</td>
</tr>
<tr>
<td>3.8 Parameter Sweep</td>
<td>29</td>
</tr>
<tr>
<td>3.9 Velvet using paired end data</td>
<td>30</td>
</tr>
<tr>
<td>3.10 A5 using paired end data</td>
<td>30</td>
</tr>
<tr>
<td>3.11 Unified paired end submission</td>
<td>30</td>
</tr>
<tr>
<td>3.12 Inferring paired end data</td>
<td>31</td>
</tr>
<tr>
<td>3.13 Data Reuse</td>
<td>33</td>
</tr>
<tr>
<td>3.14 Job Status</td>
<td>33</td>
</tr>
<tr>
<td>3.15 Plugin Wrapper</td>
<td>35</td>
</tr>
<tr>
<td>3.16 Plugin Configuration</td>
<td>35</td>
</tr>
<tr>
<td>3.17 Rest Interface</td>
<td>37</td>
</tr>
<tr>
<td>4.1 Assembly commands</td>
<td>45</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

Inherent in the tradition of scientific discovery, researchers have the natural proclivity to reduce observations to proximate or ultimate causes. Since the emergence of our curious human consciences, the questions of our provenance and resulting phenotypes have long been speculated; through Darwin to Mendel to Watson, Crick, and Franklin, it has been made clear that many of the answers lie in the ubiquitous DNA molecule. While it is certainly quixotic to envision that all of life is wholly controlled by some four-character programming language, being able to better dissect and discern patterns in this universal script has and will continue to bolster the understanding we have of its relationship to all of life’s eminence.

Indeed, it was this romantic optimism that underpinned much of the initial enthusiasm for the Human Genome Project (HGP). Announced in 1990, the HGP was a $3 billion global scientific effort to assemble the first human genome and build a map of all genes within; it was hailed as the presumable Rosetta Stone for human disease and medicine. In fact, Francis Collins, leader of the HGP, claimed that genetic diagnoses of most diseases, including cancer, diabetes, heart disease, and major mental illnesses, and their resultant treatments would be realized within 10 and 15 years’ time of the genome completion, respectively. The first draft of the human genome was released in 2000, and
declared complete in 2003. Now 10 years later, it is clear that we remain far from those idealistic assumptions. Thus far, the HGP has yielded many insights, including a better count estimate of human genes, structural differences with other organisms, as well as distributions of repeat regions, yet far from the revolution promised. In reality, the completion of the human genome may have generated more questions than it answered. Indeed, new sequencing projects are surfacing at an exponential rate in order to gain insight into genome evolution and alternate gene models; the panda genome was assembled using next-generation sequencing and recently, the Genome 10K Project was proposed, aimed at sequencing the genomes of 10,000 vertebrate species.

As sequencing technology progresses at a rate substantially faster than Moore’s Law, it has become clear that the biological sciences edges progressively closer towards becoming an information science. The biological sciences community has already become inundated with a plethora of data, of which the processing and analysis are problems caused by an overall deficiency of resources, methodological consensus, and analytical strategies, among myriad other factors.

The scope of the genome and metagenome assembly problem encompasses the fields of genetics, systems biology, graph theory, and computer architecture, among others; indeed a very interesting problem. Work is being done to progress each respective sub-discipline, such as minimizing sequencing obfuscations, classifying genetic pattern profiles, automating gene annotation, accelerating algorithm performance, or applying computational genomics to specialized hardware. However, it is becoming clear that the problem as a whole must take a comprehensive approach amongst the numerous subfields, and approaching the problem from a higher, level will undoubtedly reveal synergistic relationships in the diverse conglomeration of the field of computational genomics.
Chapter 2

Background

This chapter provides background necessary to understand the landscape and motivation of this thesis. Because the problem has its roots in multiple facets of biology, biotechnology, and computer science, understanding each sub-field intimately is crucial to our efforts of developing an optimal and integrative solution.

We discuss current sequencing technologies and the type of data generated, algorithms and methods used to process the data, and current approaches in system technology and infrastructure that are capable of handling scientific computing’s new big data problem.

2.1 Sequencing Technologies

While the new, efficient, and cost-effect next generation sequencing technologies are indeed a boon to the biology community, these benefits do not come without their inherent imperfections. Ironically enough, biologists are now producing sequencing data in record amounts, and it is this “data boom” that is one of problems faced by computational biologists today. To make matters more complex, these technologies produce reads and errors that are considerably less “assembly-friendly” than the older Sanger-based technologies. Since each platform relies on its own intricate combination of biochemistry and hardware mechanisms, distinct error profiles are associated with generated data and should be
taken into account when performing further processing. Here we survey the major next generation sequencing technologies. Overviews of major platforms are presented in Table

2.1

2.1.1 Illumina

The Illumina/Solexa GA/HiSeq systems are currently the most widely used platform in the next-generation sequencing market due to its competitive costs and high throughputs.

**Procedure**  This sequencing technology uses *sequencing by synthesis* (SBS) and *cyclic reversible termination* (SBS) and proceeds as described:

1. Single strands of the library are attached to the flowcell and bridge amplified to produce clonal clusters.

2. A DNA polymerase bound to a primed template incorporates a dye-labelled terminating nucleotide.

3. Remaining nucleotides are washed.

4. Incorporated nucleotides are detected by total internal reflection fluorescence (TIRF).

5. The terminating group and fluorescent dye are cleaved so that polymerase activity can continue.

6. Additional washing is performed and the process is repeated from step 2.

**Profile**  Initially, Solexa GenomeAnalyzer (GA) was able to output 1 Gigabase/run. Through incremental improvements in polymerase, buffer, flowcell, and software [13], the latest GAIIx series can produce 85 G/run. Introduced in 2010, Illumina’s HiSeq 2000, which employs the same strategies as above, can currently produce 100bp reads at 600 G/run, with 1 T/run possible in the foreseeable future.
The benefits of this platform are clearly throughput and cost efficacy. Error rates are relatively low as well, as after filtering they have been shown to fall below $\%2$ \[13\]. The most common error type are substitution errors, and are amplified specifically when the preceding nucleotide is guanine. Additionally, Dohm et al have shown that AT and GC rich regions are underrepresented due to amplification bias \[7\].

### 2.1.2 Roche 454

Introduced in 2005, the Roche 454 sequencing system was the first commercially successful next generation platform \[13\] and ushered in a new era of genomic studies. Wheeler et al were the first to apply next generation sequencing to personal human genomes, using 454 technology to sequence the genome of James D. Watson \[26\].

**Procedure** The 454 system employs emphpyrosequencing, which is a bioluminescence method in which enzymatic reactions cause visible light emission. The procedure is outlined here:

1. The library is denatured into single strands, captured by amplification beads, and emulsion PCR is performed.

2. The amplified targets are incubated with DNA polymerase, ATP sulfurylase, luciferase, luciferin, and adenosine 5’ phosphosulfate.

3. One of the deoxynucleoside triphosphates (dNTPs: dATPs, dTTP, dGCP, dCTP) are added, and will luminesce when a released pyrophosphate (PPi) reacts with ATP sulfurylase and luciferase.

4. Bioluminescence is detected via a charge-coupled device (CCD) camera and recorded as flowgrams. Homopolymers, for up to six base pairs, can be measured as directly proportional to signal amplitude.
5. dNTPs are degraded by apyrase and the procedure returns to step 2 using the next
dNTP type in a predetermined sequence.

Profile  The 454 GS FLX Titanium system, launched in 2008, is able to produce 0.7G/run
with reads of 700bp, and 99.9% accuracy, in under 10 hours for completion. As suggested
in the procedure, homopolymers longer than six bases produce a high error rate. Insers-
tions are the most common error, followed by deletion. Cost of reagents also remains an
issue.

2.1.3 Life/APG SOLid

Procedure  The Sequencing by Oligo Ligation Detection (SOLiD) by Applied Biosys-
tems uses an accurate two-base sequencing by ligation technology:

1. A primer is hybridized to a template amplified via emPCR

2. A “1,2-probe” is added. These consist of an interrogation dinucleotide, 16 of which
   are encoded by four fluorescent dyes, as well as additional degenerate and universal
   bases that aid in ligation.

3. the dinucleotide encoding is imaged, and the probe is partially cleaved

4. Steps 2 and 3 are repeated ten times to yield ten dinucleotide color calls in 5-base
   intervals.

5. the extended primer created thus far is stripped

6. An offset, n-1, primer is hybridized, and a second round (Step 2 - 4) ensues. DNA
   can be decoded via the knowledge of color combinations in the base interrogations.

Profile  The SOLiD 5500xl system is capable of producing 50 bp reads at 120G/run.
The highlight of the system is accuracy due to its unique dinucleotide sequence by ligation
method, at 99.94
2.1.4 Supportive Sequencing Information

One resulting strong point of next generation sequencers cost and throughput is read redundancy, or coverage. While read errors may be produced at a rate of 0.5-2.5%, using coverage and consensus, error correction can be employed to eliminate these problems. Various strategies are discussed in [section ?].

2.1.4.1 Quality/Confidence Scoring

Each sequencing technology provides a quality score for each base call traditionally by calculating a log transformed probability that the base is incorrect (phred score). Solexa scores are calculated differently, but can be approximated to the same meanings. 454 scores only indicate that a homopolymer length has been called correctly. It has been suggested that Solexa software under-estimates and over-estimates true error rates in high and low quality scores, respectively [7].

2.1.4.2 Pair Sequencing

Prior to attachment of clonal fragments to a flowcell, next-generation sequencers are able to ligate adapters to both ends, allowing for both forward and reverse reading of each strand. Furthermore, the pair of resulting reads contains positional information relative to one another. This information, “insert-size,” is leveraged in most assembly software to overcome repeat region ambiguities as well as to perform scaffolding and gap extension.

<table>
<thead>
<tr>
<th>Sequencing Strategy</th>
<th>HiSeq2000</th>
<th>MiSeq</th>
<th>454</th>
<th>SOLiD4hq</th>
<th>PacBioRS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Error Dominance</td>
<td>SBS</td>
<td>SBS</td>
<td>Pyro</td>
<td>Ligation/2base</td>
<td></td>
</tr>
<tr>
<td>Error Rate</td>
<td>0.26%</td>
<td>0.80%</td>
<td>0.01%</td>
<td>0.01%</td>
<td>12.86%</td>
</tr>
<tr>
<td>Read Length</td>
<td>2x100bp</td>
<td>2x&lt;250bp</td>
<td>2x700bp</td>
<td>2x75bp</td>
<td>1500 mean</td>
</tr>
<tr>
<td>Insert Size</td>
<td>&lt;700bp</td>
<td>&lt;700bp</td>
<td>&lt;20kb</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>Yield/Run</td>
<td>600Gb</td>
<td>2Gb</td>
<td>700Mb</td>
<td>300Gb</td>
<td>100Mbp</td>
</tr>
<tr>
<td>Time/Run</td>
<td>11d</td>
<td>39h</td>
<td>23h</td>
<td>14d</td>
<td>2h</td>
</tr>
<tr>
<td>Cost($)/Gbp</td>
<td>40</td>
<td>502</td>
<td>7000</td>
<td>70</td>
<td>2000</td>
</tr>
</tbody>
</table>

Table 2.1: Profile of Major Next Generation Sequencing Technologies
2.2 Genome Assembly

Genome assembly, or the act of combining many character sequences into, ideally, a single continuous string is an intuitively simple problem and one that is fundamental to computational genomics.

While there are multiple classes of genome assembly problems, namely reference-based (or comparison-based) and de novo assembly, the former requires a pre-assembled reference genome, which are relatively scarce especially for assembly projects of esoteric organisms, to which a mapping of newly produced reads are mapped. This is a much easier problem computationally and furthermore, de novo assembly still must be performed for larger regions where valid mappings do not exist, usually caused by genome insertions. We will concentrate on the de novo assembly problem as it poses the greatest challenges and offers immensely valuable insight.

The assembly of these reads can be formalized as a shortest common superstring computational problem:

Given a set of input sequences \([s_1, s_2, ..., s_n]\), find the shortest superstring \(C\) such that every \(s_i\) is contained within \(C\)

not only is this problem classified as NP-Complete but also convoluted by a variety of issues caused by sequencing technology and a genome’s properties itself. Next, we outline the inherent challenges as well as those caused by the imperfect sequencing technologies, and the attempts at a solution.

2.2.1 Challenges

2.2.1.1 Next Generation Sequencing

It can be said that the proliferation of new sequencing technologies, plummeting costs, and rising throughput have allowed for a drastic increase in productivity for genome sciences. But, the properties of these new technologies introduce novel challenges not seen before
with the traditional capillary sequencers. For example, the first microbial sequencing project, completed in 1995 used 24,304 reads of length 460bp. (cite)(fleischmann '95) The human genome project employed Sanger sequencing, generating roughly 30 million reads of 800bp [27] at an error rate as low as $10^{-5}$ per base(Shendure 2008). By contrast, the most widely used technology today, Illumina, generates billions of basepairs per run, at a much shorter 100bp read length and a much higher error rate. The sheer volume of data coupled with short read lengths require significant adaptations to be made in assembly methods. It has been suggested that despite that costs per basepairs has been reduced, cost per unit of information in gene annotation studies remains comparable to traditional Sanger sequencing [28].

2.2.1.2 Sequencing Error

As outlined previously, it is clear that the integrity of genomic data cannot be reliably maintained throughout the sequencing process. Each technology holds its own error profile and assembly workflows often overlook them, taking an agnostic approach in their methods. Alkan et al. [1] analyzed the differences in NCBI’s reference human genome and one assembled using next-generation sequencing. It was found that 420.2 Mbp were missing from the assembly, among various other discrepancies. A prevailing view suggests that algorithmic efficiency cannot overcome limitations introduced by sequencing technologies. Here, we describe them.

Substitution: An erroneously identified nucleotide, possibly caused by convolution in light capture signal and downstream base calling. This error is most common in Illumina and 454 platforms. Minoche et al suggest that in Illumina platforms, 99.5% of all detected errors are attributed to substitution errors. Error correction methods can use ample coverage and consensus sequences to detect errors, but low coverage yields and the detection of single nucleotide variants (SNVs) can be challenging.

Insertions and Deletions: Extra nucleotides are erroneously inserted into reads or
original nucleotides are omitted (deletion), collectively called indels. Indels are the dominant error type in 454 and IonTorrent technologies.

**Ambiguous base**: Base-calling software is unable to confidently determine a base. Although these may be completely ambiguous (thus denoted by an “N” character), software may be able to resolve base information down to base classes. (e.g. “G” or “A,” represented by the letter “Y” for “pYrimidine”). A full mapping is shown in Table 2.2.

<table>
<thead>
<tr>
<th>Code</th>
<th>Representation</th>
<th>Etymology</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>A</td>
<td>Adenosine</td>
</tr>
<tr>
<td>T</td>
<td>T/U</td>
<td>Thymidine</td>
</tr>
<tr>
<td>G</td>
<td>G</td>
<td>Guanine</td>
</tr>
<tr>
<td>C</td>
<td>C</td>
<td>Cytidine</td>
</tr>
<tr>
<td>K</td>
<td>G/T</td>
<td>KETO</td>
</tr>
<tr>
<td>M</td>
<td>A/C</td>
<td>Amino</td>
</tr>
<tr>
<td>R</td>
<td>A/G</td>
<td>Purine</td>
</tr>
<tr>
<td>Y</td>
<td>C/T</td>
<td>Pyrimidine</td>
</tr>
<tr>
<td>S</td>
<td>C/G</td>
<td>Strong</td>
</tr>
<tr>
<td>W</td>
<td>A/T</td>
<td>Weak</td>
</tr>
<tr>
<td>B</td>
<td>A</td>
<td>B after A</td>
</tr>
<tr>
<td>V</td>
<td>(T/U)</td>
<td>V after U</td>
</tr>
<tr>
<td>D</td>
<td>C</td>
<td>D after C</td>
</tr>
<tr>
<td>N</td>
<td>A/T/C/G</td>
<td>Any</td>
</tr>
<tr>
<td>-</td>
<td>Gap</td>
<td></td>
</tr>
</tbody>
</table>

Table 2.2: FastA Base Pair Codes

### 2.2.1.3 Genetic Properties

The task of reconstructing genomes is an intrinsically complex problem, and is further obscured by genome characteristics not conducive to computation. Here we describe some of these common properties.

**Repeat regions**: *Homopolymer* - Beyond causing problems for assembly, these long stretches of a single basepair cause greatly increased error rates in sequencing in
pyrosequencing platforms such as Roche/454. Even the Illumina platform, considered to be robust against homopolymers, has been observed that a length increase from 2 to 10 bases causes a 1000-fold increase of error rate [16].

2.2.1.4 Metagenomes

Metagenomics began to gain traction in 2000, when Beja et al. discovered a new ATP-generating mechanism by studying environmental fragments from seawater (Beja 00). Venter et al. later garnered attention by identifying 1.2 million genes in a single metagenomic survey of the Sargasso Sea [24] and it became increasely clear that metagenomics could be used gain insight into many functional pathways of uncultured microbial communities.

Metagenomic assembly faces the same sequencing and computational challenges as single-organism assembly, but is further complicated by poor community coverage, genome variance within natural populations, and risk of chimeric generation [22]. For instance, many error correction methods rely on read coverage and consensus for validation of an accurate read, and often times low coverage reads are discarded. An evenly high coverage of a whole metagenome is unrealistic, and furthermore, single nucleotide variants (SNVs) among a given population must not be classified as a base-calling error.

Many metagenomic studies rely on complete or draft genomes for aid in interpretation, but the relative scarcity of these remains significant. Improvements in single cell genome amplification will provide large benefits to this area of research.

2.2.2 Assembly

2.2.2.1 Overlap-Based Assembly

The most intuitive approach and accordingly first methods developed for sequence assembly are based on sequence overlaps of reads. In such overlap-based algorithms (greedy, overlap consensus), pairwise comparisons must be done amongst the reads, thus requiring
a worst case nC2 comparisons. Furthermore, the common approach for this comparison involves the Smith-Waterman algorithm, a dynamic-programming – and inherently slow – algorithm to generate an alignment or overlap score.

Alternate approaches may use heuristical methods, such as kmer similarity in TIGR, for faster performance at the expense of sensitivity. Regardless, the need to perform such comparisons and store its resulting data index causes this class of assemblers extremely computationally expensive. One advantage of overlap approaches is that it is easy to be parallelized and thus benefits greatly from appropriate multi-core computation architectures. However, with the advent of the next generation short-read, high-throughput sequencing technologies, the complexity problems inherent in pairwise methods are further exacerbated by the sheer volume of data.

### 2.2.2.2 Greedy Extension

As the name suggests, this class of assemblers uses a greedy approach, optimizing over a local objective function. In this case these assemblers optimize over the overlap scores of a defined k basepairs, or k-mers. That is, a read with the highest scoring overlap will be used for the extension. And (inherent) within these greedy, string-based algorithms is the BIG possibility of a non-optimal global solution; the resulting misassemblies are often caused by repeat regions within the genomes.

Contigs are extended with valid overlapping k-mers until the pool of potential overlaps are exhausted. Some popular assemblers that use this approach are: TIGR

### 2.2.2.3 Overlap Layout Consensus (OLC)

Overlap layout consensus methods use the same first step as greedy extension, the pairwise or heuristical overlap scoring, but instead build out a layout graph before performing the actual assembly. In this graph, the short reads are represented by nodes, and qualifying overlaps are the connecting edges. Once the graph is built, the optimal solution is the
path that traverses every node, or the Hamiltonian path. This computational problem of Hamiltonian cycles is classified as NP-Complete.

Most OLC approaches attempt to alleviate the computational requirements by employing overlap computation heuristics and performing a variety of graph reduction steps. Popular methods include exact matching, transitive reduction, collapsing chordal subgraphs, and removing dead-end paths. By using exact matching in the overlap phase, the traditional and computationally expensive Smith-Waterman scoring algorithm is bypassed while also avoiding spurious overlaps due to sequencing error. Transitive reduction involves removing branch paths in which there exists a longer path to the same node and thus are non-essential.

This proves to be a highly optimizing procedure and shown to reduce the graph complexity by a factor of the oversampling rate [17].

Graph branching is often caused by repeat regions, substitution errors, and clonal polymorphisms. (cite edena) While the first issue is unsolvable without additional information, the other problems can be addressed. Assemblers such as Edena, employ a dead-end cleaning method in which branches are traversed up to a certain threshold, and branches that fall short are removed.

Assemblers that use this OLC include Edena, CELERA, ARACHNE, Kiki, and Min-
**Algorithm 1: De Bruijn Assembly**

**Data:**

**Result:** Set $C$ of directed paths representing contigs

initialization;

forall the reads $r_i \in R$ do

  Let $\text{PREFIX}(r_i) \leftarrow$ first $k - 1$ nucleotides of $r_i$;
  Let $\text{SUFFIX}(r_i) \leftarrow$ last $k - 1$ nucleotides of $r_i$;
  Form directed edge $e_i$, representing $r_i$, from $\text{PREFIX}(r_i)$ to $\text{SUFFIX}(r_i)$;

end

Attempt to find an Eulerian path, or a path that visits every edge of the graph exactly one time. This resulting path represents the genome;

2.2.2.4 De Bruijn Graph

With the need to assemble billions of short reads from next generation sequencers, a new approach has come into favor which uses de Bruijn graphs and Eulerian paths. The method works as follows:

1. For each k-mer present within the reads, form two nodes of length k-1 corresponding to its prefix and suffix, only if unique.

2. Form a directed edge from node $x$ to $y$ if there exists a k-mer where $x$ and $y$ are its prefix and suffix, respectively.

3. Attempt to find an Eulerian path, or a path that visits every edge of the graph exactly one time. This resulting path represents the genome.

Furthermore, Euler’s theorem states that a balanced connected directed graph, which is what the constructed de Bruijn graph can be classified, must contain an Eulerian cycle.

The advantages of this method are clear: There are no pairwise comparisons for building the graph, and finding an Eulerian path is made tractable with modern compute
systems. This method corresponds fittingly to the problem, in a perfect world. Of course, previously mentioned problems inherent in sequencing and biology complicate matters, and certain strategies have been developed to best deal with these limitations.

Euler’s thereom implies that if all k-mers in the genome are generated, then there exists and Eulerian path. In the most used sequencing technology, Illumina, 100-mers generated from a genome only capture a small fraction of its source [6], thereby violating one of the assumptions of Euler’s thereom. De Bruijn assembly methods break reads in to k-mers and given small enough k, will allow for a near complete representation of the genome.

Repeat regions intrinsic to genome structure cause problems to arise in all assembly methods. In de Bruijn graphs, they cannot be represented within the graph when longer
Figure 2.2: De Bruijn Graphs. The circular genome CATTCATGTAAGTA is represented by nine reads, \{TTCAT, TCATG, TGTAA, ATGTA, ACATT, GTAAG, CATGT, AGTAC, TAAGT\}. Figure 2.1: All 3-mers in the genome are represented, but is tangled. Figure 2.2: Some 4-mers are not recovered from the reads, and thus the graph is fragmented.

2.2.3 Preprocessing

2.2.3.1 Error Correction

In Illumina sequencers, substitution, or base calling, errors occur at rates of 0.5-2.5\% \cite{11}. Other platforms such as Ion Torrent and 454, insertions and deletions due to homopolymer and carry-forward errors are common \cite{29}. These mistakes cause problems in all assembly methods, creating ambiguous or spurious paths and overlaps. Various methods to corrects
these errors have been developed. Because of the prominent usage of Illumina technology, most of these methods only target substitution errors; as new sequencing usage patterns emerge, the need to target other error profiles is becoming evident. Additionally, because final genome assembly is generally the subject of scrutiny, the comparison analysis of error correction method accuracy is often overlooked; such an elemental step requires deeper examination. Yang et al. attempt to classify the different approaches and perform a thorough evaluation. Likewise, we classify them here.

**k-spectrum** In these error correction methods, reads are first broken down into their constituent k-mers. With ample coverage and an appropriate value for k, erroneous k-mers can be inferred and corrected by measuring Hamming distances to a consensus k-mer. To explain further, a k-mer is deemed trusted if it occurs more than a given number of times, and untrusted otherwise. Untrusted k-mers are matched to a trusted k-mer if it meets a desired Hamming distance threshold, and thus corrected to conform to the consensus. Certain assemblers \([11]\), (reptile) incorporate available read quality scores into
the weighting of trusted k-mers. K-spectrum-based correction works well for substitution errors and avoids any expensive multiple sequence alignment (MSA) computation; this class is consequently popular, with substitution error being most common in Illumina reads. Popular tools using this strategy are SOAPdenovo, Quake, Reptile, SGA, and Euler-SR.

2.2.3.2 Artifact Removal

Some unfortunate side effect of high throughput sequencing are lingering non-essential artifacts due to linkers and adapters used in the initial library construction. Though removal of known library sequences from reads is a trivial computation, these sequences can undergo the same error generation as actual genome data. TagDust employs fuzzy string-matching to identify and remove true artifacts from reads [12].

2.2.4 Postprocessing

2.2.4.1 Scaffolding

Once contigs are built via assembly algorithms, the process of scaffolding attempts to place contigs in the correct order. This is generally done by using paired end reads insert size information: When a minimum number of contiguous paired end reads can be mapped and matched across two separate contigs, and where insert size is large enough to span the total sequence distance, a connection can be inferred.

Popular scaffolding tools include GRASS, SOPRA, Opera, SAKE, Bambus, and SS-PACE. Recently Bambus2 has been developed and aimed at scaffolding metagenome sequencing projects.

2.2.4.2 Gap Closing

One gap closing method seen in IMAGE and SOAPdenovo uses paired end data to extend contigs and close gaps after scaffolding has generated a supercontig. By mapping the
paired end reads back to the supercontig, the algorithm finds pairs that map one read to the contig and the other to a gap. This is done iteratively, building small islands of contiguous reads within gaps until closed or the matched-reads pool is exhausted [21]. Subsequent scaffolding may generate new gap data for additional iterations of gap closing.

[Other methods of gap closing involve comparing a collection of assemblies, perhaps generated with different assembly software or different sequencing technologies, in order to identify ways of extending contigs, merging or reconciling contigs and using contigs from one assembly to bridge gaps in another. Such methods include the graph accordance assembly (GAA) program35, Reconciliator36 and CloG37]

2.2.5 Meta-Algorithms

Many assemblers, such as ..., wrap necessary processing steps into their own internal programs to bolster the assembly. This can be restrictive and opaque to the user looking for cause, insight, or reproducibility. Conversely, a genome assembly can be approached in an integrative manner, dynamically using a collection of suitable methods or parameters to best match a provided data set. In most genome sequencing projects, this approach is taken, albeit in a very manual fashion.

2.2.5.1 A5

Developed by Tritt et al, A5 is a dynamic pipeline which integrates different tools into a multistage pipeline, and has been shown to work favorably in our testing of microbial data sets. The pipeline works as follows:

1. Ambiguous or low quality reads are removed, and the remaining are error corrected via modules in the SGA package [20]. The Tagdust [12] package is used to remove any adapter contamination.

2. The resulting reads are assembled using the IDBA assembler.
3. Contigs produced are scaffolded and extended using SSPACE.

4. Using BWA to first align reads back to resulting scaffolds, a custom method for
detection of misassemblies uses paired end data to infer “improper connections” and
breaks contigs accordingly.

5. The broken scaffolds are scaffolded once more using SSPACE.

It is interesting to note that paired end data is not initially used for the assembly via
IDBA, but only for scaffolding and breaking steps.

2.2.6 Assessment

N50 The most widely used assembly assessment metric, N50, became popular during
the completion of the human genome. N50 is calculated by sorting contigs by length,
then summing them in descending order until the total length traverses the 50% point of
the total assembly length. The length of the contig at which this happens is the resulting
score. A perhaps more representative metric is NG50, in which the decision point is at
50% of the estimated genome size. Accordingly, Nx and NGx scores can be calculated
for any value of x. An NG graph, where NG all values are calculated, is useful for a
visualized comparison of scaffold lengths relative to different assemblers as well as to
estimated genome size. [4].

Though the initial purpose of the metric was to measure the contiguity of an assembly,
it is often used to represent the quality or correctness of one. This has rightfully drawn
criticism, as any horribly misassembled contig could score just as well, if not better, than
an accurate one, given a sufficient length.

Feature Response Curve It is clear that without a proper reference assembly, a proper
assessment of a de novo assembly is essentially subjective. Narzisi et. al attempt to create
a more explanatory measurement, taking influence from Receiver Operating Characteristic
(ROC) curves to produce their Feature Response Curve (FRC) metric, which characterizes the sensitivity (coverage) of the assembly as a function of a discriminatory threshold (features). Some features include mate-pair checking, depth-of-coverage, and suggested breakpoints at suspicious regions. The generation of such profiles allows for a visual comparison between assemblers and their relative strengths and weaknesses.

**ALE**  Clark et al have developed a reference free metric called an ALE score, or Assembly Likelihood Evaluation, that is based upon a Bayesian probability statistical model [5]. By using quality data from the library from which the assembly was produced and alignment data produced from alignment tool such as BWA or Bowtie, the ALE program calculates the following subscores in order to infer the overall probability that and assembly \( S \) is generated from the set of reads \( R \), or \( P(S|R) \):

1. \( P_{\text{placement}}(R|S) \): Read quality scores and basepair alignment information, along with read orientation likelihood, are used to quantify how well the reads agree with the assembly.

2. \( P_{\text{insert}}(R|S) \): Mean insert length and standard deviation is calculated from the read mapping, and the likelihoods of each mate pair mapping is inferred.

3. \( P_{\text{depth}}(R|S) \): Given the GC content at the particular location, this measures how well the current depth agrees with expected depth inferred from GC bias models.

4. \( k \)-mer: Likelihood of the assembly \( S \) is calculated without read information by multiplying all present \( k \)-mer frequency probabilities appearing in the assembly, and used as the Bayesian prior probability.

**REAPR**  Hunt et al. have developed the “Recognition of Errors in Assemblies using Paired Reads,” or REAPR pipeline as a method to call errors in assemblies. Particularly, REAPR attempts to detect positions in the assembly at which areas between observed
and expected fragment coverage distribution (FCD) lines are greater than a calculated
threshold [9].

2.3 Computing Systems

The previous section described how next-generation sequencing has introduced numerous
algorithmic challenges to the computational biology landscape. This is, however, only
half of the issue. Beyond these conceptual problems of correctness and complexity lies
the difficulties produced by real world application. Here we explore the problems such as
data volume and computational constraints along with available technologies that may
provide solutions.

2.3.1 Scientific Compute Services

Due to the comparative nature of genomic studies, publically available repositories and
compute services are substantial for collaborative science. ...

2.3.1.1 RAST

The Rapid Annotation using Subsystems Technology (RAST) server was made available
in 2007 and is the foundation for a set of computational systems aimed at accelerating the
progress of systems biology; it is also the eponym of our Assembly RAST service. RAST
provides a fully automated service to annotate assembled contigs through identification
of protein-encoding genes, and reconstruct a metabolic network of the organism [2]. The
process of automatic annotation is based upon a knowledge base produced via manual
curation:

1. An “expert curator” defines a subsystem as a set of abstract functional roles, to
   which specific genes are connected.
2. Proteins encoded by these genes are scrutinized against a set of rules and decision procedures to create and populate protein families called FIGfams.

3. tRNA and rRNA encoding genes are identified from submitted contigs using existing tools and from these, the system intelligently infers metabolic properties of the genome.

User have the ability to submit genomes, view progress, receive notification upon completion, download results, as well as view a graphical analysis with the SEED Viewer.

2.3.1.2 MG-RAST

The Metagenomics-RAST (MG-RAST) server employs the same underlying subsystems technology of RAST to metagenomic datasets. In addition to automated functional annotations and metabolic reconstruction, MG-RAST offers additional comparison and visualization tools through summaries and subsystems heatmaps.

Due to the challenges posed by metagenomics, many traditional genome analysis methods fail to provide sufficient results. Further work to discover new methods for binning, clustering, and prediction is underway. Moreover, performance is a concern for processing of such large datasets [15].

2.3.1.3 KEGG

The Kyoto Encyclopedia of Genes and Genomes (KEGG) is a comprehensive database consisting of 15 main databases aimed at integrating genomic, chemical, and systemic functional information, along with a set of tools for interpretation of data sets [10]. Via the KEGG Mapper, a user is able to submit queries of genes, proteins, and other molecular objects and attain pathway and ontology enrichments. KEGG offers several other tools, such as those for genome metabolic comparison, and drug and disease information. As of 2011, access to the KEGG FTP knowledge base has moved to a paid subscription model.
Chapter 3

Implementation

3.1 Introduction

The Assembly RAST system is an open-source framework based on a variety of technologies that allows for workflow flexibility, extensibility, and automation, among other things, in order to abstract away the complexities of numerous unstandardized bioinformatics tools currently available.

Unlike current multi-stage pipeline frameworks, this approach characterizes and modularizes pipeline steps, and offers the capability to either explicitly piece together a custom meta-algorithm pipeline, or to intelligently infer the optimal order, processing, reprocessing, or parameter thresholds for an assembly pipeline.

By allowing the user to direct focus upon experimentation of various methodologies, algorithms, and meta-algorithms, as well as providing access to a variety of post-assembly analyses and visualization tools, better outward approaches to discovering optimal strategies can be explored.

By using a service-oriented architecture, the Assembly RAST system offers a high level of scalability and extensibility. As the genome assembly problem is computationally expensive, and metagenomes further exacerbate this landscape, it is important that a
system built to tackle such a problem be ready to handle impending data booms. Not only is the Assembly RAST system already bolstered by several high-powered compute engines, but attaching additional workers is seamless. Furthermore, as computation problems or subproblems become better solvable on specialized hardware, the system’s plugin framework can handle nearly any esoteric workflow.

The overall system framework is implemented using Python and various open source technologies including RabbitMQ, MongoDB, CherryPy, and OpenStack. In this chapter, we describe the functionality of the framework pertaining to genome assembly and why it is conducive to experimentation, and proceed to give an overview of the system infrastructure and its role as a service.

3.2 Assembly Pipelines

While raw sequence reads can be assembled solely by using singlar de novo assembly tools, many research efforts have shown optimal results when a multi-stage processing pipeline is used. Generally, this workflow involves filtering, error correction, assembly, scaffolding, and post-processing. Thus, one of the main design tenets of the system is the ability to easily invoke or generate intricate pipelines in user-friendly and relatively simplified

3.2.1 Unified Assembly Invocation

Bioinformatics tools and and analysis methods are becoming abundant, but at the cost of proliferative commandline inconsistencies and non-traditional UNIX commandline invocations. Generally, for a particular tool class, the same required inputs are needed for invocation. For example, and assembler commandline binary usually asks for the read file(s), and other optional flags. As simple as this may be, we see a buckshot of esoteric commands.
Figure 3.1: Assembly and Analysis Workflow

Listing 3.1: Velvet Command

```
velveth outpath 29 -fastq -shortpaired1 reads1.fq reads2.fq
velvetg outpath
```

Listing 3.2: Kiki Command

```
kiki -i reads.fa -o outputfile -k 29
```

Listing 3.3: IDBA Command

```
# Merge and convert to FastA
fq2fa --merge --filter reads1.fq reads2.fq reads.fa
idba -r reads.fa -o ~/output --maxk 32
```

Assembly RAST aims to simplify this by implementing tools as plugins with flag
defaults, where optional overrides are available.

27
While there are multiple entry points to invoke the assembly RAST service, as discussed in CHAPTERX, example commands are shown using the commandline client ARAST.

Listing 3.4: Example Invocation of Assemblers

```
arast run --f reads.fa --assemblers kiki velvet idba
```

### 3.2.2 Explicit Pipeline Invocation

Traditionally, genome assemblies are completed in a precise but manual manner, invoking certain tools in a linear fashion from the commandline. Depending on the level of scripting, this may impose unnecessary latencies due to users having to monitor a stage’s progress and invoke proceeding steps by hand. Furthermore, this practice of invoking from the commandline is anticonducive to maintaining a reproducible procedure. There are, however, ongoing efforts to create more advanced and cohesive scripted pipelines, with good results.

Currently available genome assembly pipelines, such as A5 or JigSaw, are predeter-
mined scripts in which the user has little choice or flexibility over settings or pipeline modules. Tweaking parameters or pipeline stages involves manually editing the pipeline scripts. Furthermore, attempts to hypothesize and find optimal parameter settings or pipeline configurations would involve multiple copies/alterations of these scripts. With the Assembly RAST pipeline invocation language, the user can call upon a simplified yet flexible string to generate the desired pipeline or pipelines.

Listing 3.5: Example Pipeline

```
arast run -f reads.fa --pipeline trim_sort tagdust velvet
```

Listing 3.6: Example Pipeline with Explicit Parameters

```
arast run -f reads.fa --pipeline trim_sort tagdust velvet
```
Often times, it is unclear how a certain parameter configuration or pipeline stage affects a resulting assembly. ARAST supports the generation of pipelines in which modules can be easily substituted or run with different configurations. Data is reused when possible.

**Listing 3.7: Pipeline Branching**

```bash
arast run -f reads.fa --pipeline "none trim_sort" "idba velvet"
```

(Example of pipeline branching, data reuse)

### 3.2.3 Automatic Pipeline Branching

Certain assembly tools such as IDBA or Velvet optimiser employ an iterative parameter approach with beneficial results. While these tools generally iterate over a single parameter such as k-mer length [VERIFY], ARAST is built on a framework that allows for more universal parameter SWEEPS and optimization.

**Listing 3.8: Parameter Sweep**

```bash
arast run -f reads.fa --pipeline 'kiki ?k=29-34'
```

### 3.2.4 Automatic and Dynamic Optimization

Many approaches take a multistep optimization approach to assembly, employing parameter sweep iterations in order to capture the best features resulting from specific settings. IDBA iterates over a range of k-values, and VelvetOptimiser optimizes the Velvet assemblers over k, expected coverage, and coverage cutoff settings. However, these optimization functions are built from assessment of contig size; it is clear that complete reliance upon
N50 for evaluation and scoring is unwise. We find it important to consider various evaluation scoring techniques, and designed the analysis framework to remain agnostic and flexible. The current implementation features the ability to return an “optimal” assembly from a set of pipelines. It is in our future interest to implement a search meta-algorithm that reduces unnecessary parameter iterations.

3.2.5 File Handling

Read data generated from sequencers are ordinarily stored in the FastA and FastQ formats, the latter containing quality information. Due to the plain text ASCII format, data is often compressed with common compression algorithms. Automatic detection and decompression of these files are supported on the compute side, alleviating a fair amount of network load if used.

Paired end data is essential to many genome processing tools. However, syntax for conveying pair library data for invocation remains highly variable.

Listing 3.9: Velvet using paired end data

(velvet paired)

Listing 3.10: A5 using paired end data

(a5 library config file)

AssemblyRAST is able to correctly handle and appropriate explicit paired end library instruction to receiving modules. Also, the service partially supports paired end library inference base on file names.

Listing 3.11: Unified paired end submission

arast run --pair read1.fa read2.fa ?ins=300 --assemblers velvet
3.3 Visualization and Rapid Analysis

Automatic and rapid analysis within the Assembly RAST workflow is necessary on two fronts. First, an intra-pipeline ONGOING analyses and calculations are necessary for the meta-algorithm controlling the overall stages of the pipeline, mentioned previously. And second, a thorough statistical analysis of pipeline results are returned to the user alongside visualizations generated from the data. This will allow users quick access to the necessary metrics that will give insight into the quality of the

3.4 System Design and Infrastructure

3.4.1 System Overview

The AssemblyRAST infrastructure consists of five separate components, all of which could exist on one machine, or many:

Control server  a RESTful frontend listens for client calls, performs authentication checks, analyzes request, populates job queue, hosts web frameworks.

Data repository  uses Shock technology, permanently stores raw data, computed data, and userspace files.

Job queue  manages different work queues in which job distribution is determined by rules pertaining to worker and job classifications

Compute worker systems(s)  various system types running the AssemblyRAST compute framework
Client various entry points into the AssemblyRAST system including command line interface (CLI), RESTful API, KBase Iris, and web UI.

3.4.2 Control Server

The control server acts as the main entry point for any actions or requests to the AssemblyRAST system. The server employs the use of CherryPy, a simple web framework for Python, to implement a RESTful interface.
3.4.2.1 Request Interface

Jobs can thus be submitted in a variety of ways, and data transfer is handled by the UNIX `curl` and wrapper libraries. Along with job and data submission, job status can be queried and the transfer of computation results to the user is handled by the server.

3.4.2.2 Job Delegation

When a job request is submitted, the server creates a database record, determines the queue in which it will be placed based off of details such as data size, data format, organism type, or explicit request parameters. This type of job delegation is important for the imminent future, when the need to handle thousands or microbial genomes as well as massive metagenome becomes apparent.

3.4.2.3 User Space

While the AssemblyRAST server is open to the public, we use Globus Online’s Nexus OAuth2 API for authentication, as it offers a robust solution for user registration and enables the creation of unique userspaces for data and record keeping. Network speed is yet another bottleneck to such a service; a userspace allows data reuse if the user desires to repeat an assembly job or attempt an alternate pipeline approach.

```
Listing 3.13: Data Reuse

arast run --data 42 --assemblers kiki velvet
```

```
Listing 3.14: Job Status

+--------+---------+--------------------+----------+-------------+
| Job ID | Data ID | Status            | Run time | Description |
+--------+---------+--------------------+----------+-------------+
| 83     | 40      | pipeline [success]| 0:22:29  | None        |
| 92     | 40      | Running: [4%]     | 0:00:22  | pch1        |
```
3.4.2.4 System and Cloud Monitoring

By monitoring system loads and queue sizes, the system attempts to balance job load by launching additional OpenStack VMs as needed.

3.4.3 Compute Runtime

The compute runtime is designed to function as a redundant standalone module of the AssemblyRAST system to be deployed upon varying system types and architectures. As we continue to explore the performance suitability of different hardware profiles, the flexibility to assign specific workflows based on compute systems is key to maximizing computational resource efficiency. Currently, numerous big-memory compute VMs are deployed to process standard assembly jobs, and implementation on specialized hardware such as Convey FPGA-accelerated hybrid-core servers and NERSC supercomputing clusters is undergoing development for sequence alignment and metagenomic assembly computation, respectively. Volunteer computing is also a possible future target.

3.4.3.1 Queue Listening

Because each machine may have unique properties, initial configuration allows the particular compute worker to “subscribe” to specific queues, or queue categories.

3.4.3.2 Plugin Framework

The submodular functionality of the compute runtime is dependent on core bioinformatics tools designed to run directly in a CLI-to-Unix-process manner. A plugin framework is
thus necessary to abstract away the variability seen in these component invocations and function harmoniously with the meta-algorithmic workflows in our computations.

Listing 3.15: Plugin Wrapper

```python
class KikiAssembler(BaseAssembler, IPlugin):
    def run(self, reads):
        cmd_args = [self.executable, '-k', self.k,
                    '-o', self.outpath + '/kiki', '-i']
        cmd_args += self.get_files(reads)
        self.arast_popen(cmd_args)
        return glob.glob(self.outpath + '/*.contig')
```

The framework features a plugin class, and is further extended into different sub-classes for assemblers, preprocessing tools, scaffolders, and possibly other categories for alternate behaviors. By inserting modules into the plugin interfaces, output files, logging, benchmarking, and statistics can be automatically handled for further use. Module properties and default command line flags or parameters can be easily set within the plugin configuration file.

Listing 3.16: Plugin Configuration

```
[Core]
Name = kiki
Module = kiki

[Settings]
executable = /usr/bin/ki
filetypes = fasta, fa, fastq, fq
k = 29
contig_threshold = 1000
```
3.4.3.3 Data and Efficiency

Sequencing data is commonly produced as a relatively inefficient plain text format. This, coupled with the high-throughput is cause for concern in storage space as well as data I/O within the workflow itself. As noted earlier, users have the ability to reuse previously uploaded data, as it is permanently stored in the data repository. The compute runtime also performs checks in attempt to locate the requested data on the local node itself, in the case that it was the same worker that handled the particular data set in prior jobs. If no cached copy exists on the node, it is transferred accordingly.

A large fraction of running time for bioinformatics processing programs is attributed to disk I/O. Furthermore, passing intermediate processing states between fundamentally separate binaries cannot be solved easily, especially in the case where an analytical scoring of a particular state is necessary for the invocation parameters of the next stage. For example, a meta-algorithm’s invocation of a gap-closing stage is dependent upon the n50 score of an assemblers contig output. An N50 score requires a completely finished and sorted list of contigs, and storing all data in memory may not be feasible; thus each intermediate state must involve disk I/O.

Because of the iterative and sometimes exhaustive workflows generated by the service, data efficiency is paramount. Thus, intermediate states are reused when possible.

Currently, RAM drives are being tested and may offer a boost in throughput, though developing a more intraprocess approach is in the interest of future work.

Finally, workflow disk space requirements are predicted and a garbage collection agent is employed to ensure sufficient space.

3.4.4 Workload, Robustness, and Scalability

The assembly, processing, and analysis of genomes is a computationally expensive routine that requires considerable resources and time. Thus, scalability as well as a robust queueing system are necessary features in such a system.
Assembly RAST features a compute node monitor as well as a queuing system that employs RabbitMQ, a framework that implements the Advanced Message Queuing Protocol (AMQP). By monitoring the overall system load and queue size, an adequate number of VMs can be ensured as new compute images can be launched and terminated within the local OpenStack cloud.

3.4.5 Data

As with any scientific experiment, procedures of genome assemblies must be reproducible. Moreover, with the frailties of software designs and patchy script workflows prominent in assembly pipelines, the practice of bookkeeping is paramount for an accurate genome assembly verification and further analysis. And finally, the extrinsic analysis of the hardware performance requires additional intra-computational benchmarking and data collection.

3.5 User capability

3.5.1 RESTful API

Listing 3.17: Rest Interface

```
curl -X GET http://www.kbase.us/service/assembly/user/johndoe/job/42
```
Chapter 4

Results and Analysis

The AssemblyRAST service has been fully developed, undergone deployment and robustness tests by the KBase development team and as of February 2013, is released as a “KBase Labs” module, freely available for use by the general public. The service has been used to assemble over 100 USDA Brucella genomes to be annotated by the RAST system. The scalable infrastructure allows for high throughput for assembly pipelines, and gives users an ideal environment for the study of assembly algorithms at a larger scale than conventional desktop machine experiments. Here we present a few examinations of de novo assembly pipelines and methods. Figures 4.1 4.2 and 4.3 offer an example of the types of visualizations generated by the AssemblyRAST server.

4.1 Example Assemblies

Given the prolific number of assembler tools currently available, it is necessary to evaluate and compare different approaches against varying datasets. While other representative studies compared the assemblies of various assemblers [14], we additionally tested various pipeline and parameter configurations to evaluate the efficacies of different processing stages, such as quality trimming, error correction, misassembly detection, and gap closing, in order to develop a better understanding of their relationships. Furthermore, we explored
4.1.1 Multiple Assembler Comparison

In attempts to answer the question of which standalone assembly tool produces the best assemblies from a single shotgun sequencing library, we chose NUM data sets which had an existing reference genome, and NUM data sets which did not, ranging in length from ... to ... These sets were produced using either Illumina HiSeq or MiSeq technologies, allowing for an additional evaluation of how assemblers responded to these distinct sequencing methods.

4.1.1.1 Data

The four data sets with corresponding reference genomes were the same used in the GAGE-B [14] study and thus available at the studies website (http://ccb.jhu.edu/gage_b/): B. cereus MiSeq from the Illumina website, Reference: B. cereus ATCC
Data Cleaning

Typically, raw data from sequencers contains low quality sequences, contaminants, and adapter sequencers that should be discarded. A variety of tools that
filter and trim these types of reads are available, and we employed some on the data
data sets to improve assemblies. For the HiSeq data sets of V. cholera and R. sphaeroides, we
used the available trimmed set from GAGE-B. Magoc et al. used the ea-utils package to
remove adapters and perform q10 quality trimming. For the S. aureus data set, we used
raw data. We found that the MaSuRCA, IDBA-UD, and accordingly A5 became unstable
for data sets with highly variable read lengths, characteristic of MiSeq data. Thus for the
B. cereus MiSeq data set, in addition to quality trimming and adapter removal, we filtered
out any reads that were not within the range of 150-200 base pairs in length.

4.1.1.2 Evaluation Metrics

To determine accuracy of assemblies, the following metrics were measured:

Number of contigs : The number of contigs assembled over 500bp long.

N50 : As described in section 2.2.6 the length of the shortest contig at which 50% of
the total assembly length is contained in all contigs larger than the N50 contig.
Nx: Analogous to N50, but instead uses x% in the calculation. This is useful for graph visualization across all values of x.

For data sets with an available reference genome, structural variation metrics were able to be measured:

**Misassemblies**: Using the reference genome for comparison, we sum up the total number of relocations, translocations, and inversions detected. For a contig \( C \), misassembly point \( m \), and left and right flanking sequences \( L_m \) and \( R_m \), relocation misassembly breakpoint occurs at \( m \) when \( L_m \) and \( R_m \) align on the same reference chromosome, but over 1000bp away, or overlap by 1000bp. Inversions are defined as errors in which the \( L_m \) and \( R_m \) do not qualify as a relocation, and align to opposite strands of the chromosome. Finally, a translocation occurs when \( L_m \) and \( R_m \) align to different chromosomes.

**Substitutions**: Mismatched base pairs in all alignments.

**Indels**: Insertions or deletions in all alignments.

**NGA50**: A “corrected” version of N50, where contigs are broken at alignment misassemblies into blocks, and block length is used instead of contig length.

**NGAx**: Analogous to NGAx, where x% is instead used in the calculation.

These metrics were calculated using QUAST 2.1 [8]. For data sets without an available reference genome, reference-free assessment methods were used:

**ALE Likelihood**: Likelihood of an assembly given the initial read library, as described in section 2.2.6

**REAPR Misassemblies**: An error calling method which measures the difference in observed and expected fragment coverage distribution at each base pair, as described in 2.2.6
4.1.1.3 Assemblers

The following assemblers were used in the comparison:

- **Velvet** [30]: DeBruijn graph assembler designed for short read sequencing technologies.

- **Kiki** (Xia et al. 2012)

- **IDBA-UD** [19] - Iterative DeBruijn graph Assembler: Iterates over k-mer sizes to capture strengths in small and large parameters.

- **SPAdes** [3]: Features a multi-deBruijn graph approach. SPAdes typically combines a preprocessing step using the BayesHammer error correction tools and a misassembly detection postprocessing tool. For these assemblies, we isolated the SPAdes assembly step.

- **MaSuRCA** - Maryland Super Read Cabog Assembler: Uses a combination of both overlap consensus (OLC) a deBruijn graph assembly algorithms.

- **A5** [23]: a pipeline which uses a mixture of open source tools, their own algorithms, as well as dynamic parameter configuration. Namely, it uses preprocessing and error correction from SGA [20], the IDBA-UD assembler, and SSPACE to scaffold. For the purpose of these comparisons, we use contigs before scaffolding.

- **A6**: We modified the A5 pipeline to filter read lengths and ensured proper quality encoding detection for stability.

4.1.1.4 Comparisons

While the most often used metric for assembly quality is N50, it can be misleading, as an assembly containing many misassemblies will produce an inflated N50 score. For assemblies with a corresponding reference genome, a more accurate NGA50 score can be
Table 4.1: Comparison of NGA50 scores. Best values are shown in bold. NGA50 score is described in section (*A6 was used.

<table>
<thead>
<tr>
<th>Organism</th>
<th>MaSuRCA</th>
<th>Kiki</th>
<th>Velvet</th>
<th>IDBA</th>
<th>SPAdes</th>
<th>A5</th>
</tr>
</thead>
<tbody>
<tr>
<td>B. cereus HiSeq*</td>
<td>52644</td>
<td>59995</td>
<td>42763</td>
<td>31347</td>
<td><strong>78420</strong></td>
<td>45935</td>
</tr>
<tr>
<td>S. aureus</td>
<td>22603</td>
<td>1854</td>
<td>11540</td>
<td>34957</td>
<td><strong>50888</strong></td>
<td>8188</td>
</tr>
<tr>
<td>V. cholera HiSeq</td>
<td>59028</td>
<td>42804</td>
<td>47191</td>
<td>70796</td>
<td><strong>177768</strong></td>
<td>72282</td>
</tr>
<tr>
<td>V. cholera MiSeq</td>
<td>50207</td>
<td>70738</td>
<td>19767</td>
<td>44178</td>
<td>198488</td>
<td>57376</td>
</tr>
<tr>
<td>R. sphaeroides HiSeq</td>
<td>66418</td>
<td>3893</td>
<td>33342</td>
<td>72357</td>
<td>71175</td>
<td>20356*</td>
</tr>
<tr>
<td>R. sphaeroides MiSeq</td>
<td>-</td>
<td>33589</td>
<td>62923</td>
<td>60228</td>
<td><strong>126502</strong></td>
<td>83693</td>
</tr>
</tbody>
</table>

Table 4.2: Comparison of misassemblies

produced, which measures aligned blocks that are broken at misassemblies, rather than contig length, and normalized over the length of the reference genome. We summarize the NGA50 scores and errors of all assemblies in Tables 4.1 and 4.2. For the B. cereus HiSeq data set, the reference was found to be too divergent by mapping, so the N50 scores of its assemblies are reported instead. For all assemblers, default or “auto” settings were used, except for Velvet, for which we performed \( k \)-mer searches on most of the datasets. Where applicable, the best assembly score was chosen, and hash length \( k \) is reported in Table 4.6. Three of the data sets, B. cereus MiSeq and R. sphaeroides HiSeq and MiSeq, matched the reference genomes precisely, and while the V. cholera and S. aureus reference genomes were of a different strain, they were inferred to be similar enough to use for the data sets. Errors detected in the latter sets may represent true variation.

For the S. aureus HiSeq assemblies, SPAdes generated the largest NGA50 score, followed by IDBA-UD and MaSuRCA. IDBA-UD, however, contained the least amount of misassemblies, and SPAdes contained one of the highest amounts, mainly relocations as described in Section 4.1.1.2.

For the V. cholera HiSeq assemblies, SPAdes generated the largest NGA50 score at
177kb, more than doubling the next largest score of Velvet at 79kb. IDBA-UD, A5, and SPAdes contained few misassemblies at 4, 5, and 7, respectively. Kiki and MaSuRCA contained a relatively high amount of misassemblies at 59 and 60, respectively. All contigs generated covered at least 94% of the genome. Similarly for the MiSeq dataset, SPAdes performed the best with 198kb, followed by Kiki at 70kb.

For the R. sphaeroides HiSeq assemblies, IDBA-UD generated the largest NGA50 score at 72kbp, followed closely by SPAdes at 70kbp. Accordingly, IDBA-UD produced the fewest misassemblies.

### 4.1.1.5 Commands

This section lists the commands generated by AssemblyRAST.

Listing 4.1: Assembly commands

```bash
# Command given to AssemblyRast client:
```

45
Figure 4.5: NGAx plot of V. cholera assembly

```
ar_run --pair frag_1.fastq frag_2.fastq insert=180 stdev=20 \ 
    -a masurca kiki velvet idba spades a5
============ Pipeline 1: [masurca] =============
runSRCA.pl config.txt
bash assemble.sh

# Contents of config.txt:
PATHS
CA_PATH = /home/ubuntu/assembly/bin/MaSuRCA-2.0.0/CA/Linux-amd64/bin/
JELLYFISH_PATH = /home/ubuntu/assembly/bin/MaSuRCA-2.0.0/bin/
SR_PATH = /home/ubuntu/assembly/bin/MaSuRCA-2.0.0/bin/
END

PARAMETERS
GRAPH_KMER_SIZE = auto
USE_LINKING_MATES = 1
KMER_COUNT_THRESHOLD = 1
```
<table>
<thead>
<tr>
<th>Assembly</th>
<th>MaSurca</th>
<th>Kiki</th>
<th>Velvet</th>
<th>IDBA-UD</th>
<th>SPAdes</th>
</tr>
</thead>
<tbody>
<tr>
<td># contigs (&gt; 0 bp)</td>
<td>135</td>
<td>1228</td>
<td>733</td>
<td>301</td>
<td>316</td>
</tr>
<tr>
<td># contigs ( = 1000 bp)</td>
<td>125</td>
<td>1228</td>
<td>520</td>
<td>140</td>
<td>133</td>
</tr>
<tr>
<td>Total length ( &gt; 0 bp)</td>
<td>4454671</td>
<td>4211242</td>
<td>4604732</td>
<td>4550407</td>
<td>4661794</td>
</tr>
<tr>
<td>Total length ( = 1000 bp)</td>
<td>4447475</td>
<td>4211242</td>
<td>4523850</td>
<td>4490529</td>
<td>4604439</td>
</tr>
<tr>
<td># contigs</td>
<td>133</td>
<td>1228</td>
<td>587</td>
<td>173</td>
<td>161</td>
</tr>
<tr>
<td>Largest contig</td>
<td>159677</td>
<td>33373</td>
<td>56029</td>
<td>378946</td>
<td>230007</td>
</tr>
<tr>
<td>Total length</td>
<td>4453816</td>
<td>4211242</td>
<td>4572094</td>
<td>4512010</td>
<td>4625338</td>
</tr>
<tr>
<td>Reference length</td>
<td>4603060</td>
<td>4603060</td>
<td>4603060</td>
<td>4603060</td>
<td>4603060</td>
</tr>
<tr>
<td>GC (%)</td>
<td>68.86</td>
<td>68.55</td>
<td>68.68</td>
<td>68.81</td>
<td>68.82</td>
</tr>
<tr>
<td>Reference GC (%)</td>
<td>68.79</td>
<td>68.79</td>
<td>68.79</td>
<td>68.79</td>
<td>68.79</td>
</tr>
<tr>
<td>N50</td>
<td>74831</td>
<td>4406</td>
<td>14135</td>
<td>73097</td>
<td>71177</td>
</tr>
<tr>
<td>NG50</td>
<td>66418</td>
<td>3936</td>
<td>13982</td>
<td>72396</td>
<td>73118</td>
</tr>
<tr>
<td>N75</td>
<td>34138</td>
<td>2548</td>
<td>7489</td>
<td>42086</td>
<td>47310</td>
</tr>
<tr>
<td>NG75</td>
<td>31030</td>
<td>2088</td>
<td>7240</td>
<td>41189</td>
<td>47310</td>
</tr>
<tr>
<td># misassemblies</td>
<td>7</td>
<td>32</td>
<td>14</td>
<td>3</td>
<td>9</td>
</tr>
<tr>
<td># local misassemblies</td>
<td>12</td>
<td>5</td>
<td>1322</td>
<td>4</td>
<td>7</td>
</tr>
<tr>
<td>Unaligned contigs length</td>
<td>0</td>
<td>2760</td>
<td>117</td>
<td>0</td>
<td>55</td>
</tr>
<tr>
<td>Genome fraction (%)</td>
<td>95.315</td>
<td>90.776</td>
<td>95.681</td>
<td>97.712</td>
<td>99.048</td>
</tr>
<tr>
<td>Duplication ratio</td>
<td>1.016</td>
<td>1.006</td>
<td>1.036</td>
<td>1.005</td>
<td>1.017</td>
</tr>
<tr>
<td># N's per 100 kbp</td>
<td>0.00</td>
<td>0.00</td>
<td>3294.32</td>
<td>0.00</td>
<td>8.76</td>
</tr>
<tr>
<td># mismatches per 100 kbp</td>
<td>31.34</td>
<td>32.95</td>
<td>8.54</td>
<td>4.16</td>
<td>12.11</td>
</tr>
<tr>
<td># indels per 100 kbp</td>
<td>5.54</td>
<td>6.99</td>
<td>23.91</td>
<td>3.58</td>
<td>5.75</td>
</tr>
<tr>
<td>Largest alignment</td>
<td>159677</td>
<td>33373</td>
<td>53699</td>
<td>378908</td>
<td>230007</td>
</tr>
<tr>
<td>NA50</td>
<td>74744</td>
<td>4285</td>
<td>12944</td>
<td>73097</td>
<td>67626</td>
</tr>
<tr>
<td>NGA50</td>
<td>66418</td>
<td>3893</td>
<td>12905</td>
<td>72357</td>
<td>71175</td>
</tr>
<tr>
<td>NA75</td>
<td>34138</td>
<td>2483</td>
<td>6437</td>
<td>41275</td>
<td>42056</td>
</tr>
<tr>
<td>NGA75</td>
<td>31030</td>
<td>2048</td>
<td>6263</td>
<td>37563</td>
<td>42056</td>
</tr>
</tbody>
</table>

Table 4.3: All statistics of R. spaeroides assemblies

LIMIT_JUMP_COVERAGE = 60
NUM_THREADS = 8
JF_SIZE = 2000000000
EXTEND_JUMP_READS = 0
CA_PARAMETERS = ovlMerSize=30 cgwErrorRate=0.25 ovlMemory=4GB
DO_HOMOPOLYMER_TRIM = 0
END

DATA
PE: p1 180 20 frag_1.fastq frag_2.fastq
END
4.1.1.6 Discussion

The data reported in Tables 4.1, 4.2 and 4.3 sheds some light on the overall performances of the standalone assembly tools. We found that the SPAdes assembler generated the largest NGA50 scores for ... of the datasets, followed by IDBA-UD, which generated the largest NGA50 score for ... of the datasets. Both of these assemblers employ a multisized $k$-mer approach, so it would appear that such a strategy should be built upon for future iterations of assemblers. One point to note, however, is that SPAdes does incorporate ambiguous base pairs, or “N’s” into assembled contigs, where all others will not concatenate such uncertainties. While these incidents are fairly low in comparison to scaffolding steps, it is currently unclear as to how this may affect such contiguity scores such as N50 and NGA50. MaSuRCA was shown to be a promising assembly method by Magoc et al., but for our experiments, the software was unstable using default or auto settings. Further exploration must be performed, as this assembler’s potential has not yet been fully utilized.
4.1.2 Read and Assembly Processing Stages

In addition to many assembly tools, the computational biology community has created a diverse set of standalone preprocessing and post-assembly tools in hopes to improve both the act of assembly as well as the correction or “polish” of an assembly produced. While many assemblers come packaged or hard-coded with these pre- and post steps, we began to consider various combinations of tools for better insight into how differing approaches to processing perform when coupled with alternative methods of assembly.

We investigated the following processing steps by invoking pipelines of the V. cholerae HiSeq dataset:

- Ea-Utils Q10 Trimming (Trim)
- SGA Preprocessor (SGAp)
- SGA Error Correction
- Bayes Hammer Error Correction (BH)
- Tagdust adapter removal (TD)
- Read length filtering

For assemblies using the Ea-Utils trimming step, we used previously processed data from GAGE-B. The NGA50 and misassembly performances for various combinations are shown in Table 4.4 and Table 4.5. For this dataset, SGA error correction had little to no affect on downstream assembly, so it was omitted from the table. For the IDBA-UD assembly, the raw data set produced the largest NGA50 score at 72kbp as well as the lowest misassembly rate at 3. Notably, it performed the worst when using data corrected by BayesHammer, while other processing steps only performed slightly worse than raw data.
MaSuRCA was unable to complete some of our pipeline configurations, but for those that completed, it seemed generally unaffected by the varying combinations. Kiki performed well using a combination of SGA preprocessing, BayesHammer, and Tagdust at NGA50 of 47kbp, more than 12 times larger than its assembly of raw data. Finally, SPAdes generates an NGA50 score nearly three times larger with q10 trimming and Tagdust than with raw data.

4.1.2.1 Discussion

Because many assemblers are written as miniature pipelines in which reads are processed, controlled for quality, and/or error corrected, it remains difficult to form conclusions as to which processing pipelines may produce optimal results. This is clear when considering the performance of IDBA-UD on raw data. The IDBA-UD program employs an internal error correction mechanism which cannot be isolated from the assembly steps. It would be worthwhile to investigate this iterative k-mer approach decoupled from error correction. In our list of tested assemblers, Kiki can be viewed as a “true” assembler in the sense that it only the graph assembly and correction steps with no prior manipulation to the reads. This explains the massive improvement when using quality control and error correction prior to the assembly stage. By default, SPAdes is coupled with the BayesHammer error correction stage; we disabled the step in some of the trials. Interestingly, SPAdes perform better, albeit slightly, without the use of BayesHammer.

4.1.3 Parameter Search

Most available assembly tools offer the ability to set various initial parameters, such as k-mer length for DeBruijn graph algorithms. While many assembler comparison studies attempt to assess assembly quality across different assembler programs, the effect of
different parameters on assemblies remains largely unexamined. Here, we used the AssemblyRAST system to perform parameter sweeps and attempt to discern their relationships with data set properties and assembly quality.

We used six data sets of three organisms: B. cereus HiSeq and MiSeq, R. sphaeroides HiSeq and MiSeq, and V. cholerae HiSeq and MiSeq. By using two sequencing technology data sets per one reference genome, and since these data set pairs show slightly different read profiles we attempt to find a relationship between initial raw data profiles and optimal hashlength size in Velvet. Various read data statistics, along with optimal hash lengths and largest NGA50 scores are shown in Table 4.6.

For each data set, we performed Velvet assemblies where we set Velvet’s “hash_length” parameter to a range of 29 to 65 with a step size of 4. A plot of hash_length versus the reference-based NGA50 alignment metric is shown in Figure 4.7. The B. cereus HiSeq
The data presented in Table 4.6 and Figure 4.7 offer some insight into the effects of Velvet’s de Bruijn graph $k$-mer hashing size on assembly quality. First, when considering the $k$-mer sweeps of each data set individually, all show either monotonic trends or local maxima. Thus, larger $k$-mer sizes must be explored for the monotonically increasing experiments, then with regard to a certain criteria, an optimizing function could
Figure 4.7: Velvet Hash Length vs. NGA50 Score on Rsp HiSeq and MiSeq

be implemented to give the best assembly. With a reference genome available, NGA50 score would be ideal for this criteria; without a reference, however, the defining criteria remains equivocal. Next, examining trends of each sequencing technology offers no conclusive relationships between the k-mer sizes and the profiles of the datasets. The HiSeq datasets contained reads averaging from 78 basepairs to 98 basepairs, roughly half of the MiSeq datasets’ 152 basepair to 252 average range. Furthermore, MiSeq data assembled better in comparison to their respective HiSeq data for only two of the organisms, B. cereus and R. spaeroides, and vice versa for V. cholerae. Further studies are thus required to gain insight into how best to choose graph-building parameters.

4.1.4 Hybrid Assembly Merging

As shown in prior sections, assembler performance appears to vary across data sets, as some assemblers are able to capture portions of an organism’s genome from the reads, while the same ones show weakness on other genomes. Here, we explore the idea of using
final assemblies from multiple assemblers by merging.

We investigate merging by using Genomic Assemblies Merger for Next Generation Sequencing (GAM-NGS), which uses read alignments to identify similar “blocks” within each assembly \[25\]. We performed assemblies on the V. cholerae dataset using the assemblers Kiki, Velvet, IDBA-UD, and SPAdes, and performed pairwise mergings. NGA50 scores for each merger are shown in the Figure 4.8 heatmap. Besides two mergers in which SPAdes was the “master” assembly, all mergers showed improvement in NGA50 scores when merged with another assembly. While merging the SPAdes assembly with Kiki’s or Velvets yielded no improvement, the SPAdes-IDBA merger had the largest overall NGA50 score at 198kbp.

![NGA50 scores of pairwise mergings of V. Cholerae assemblies using GAM-NGS](image)

Figure 4.8: NGA50 scores of pairwise mergings of V. Cholerae assemblies using GAM-NGS. The principle diagonal represents assemblies before merger.

### 4.1.4.1 Discussion

Like the multi-deBruijn graph approaches of IDBA and SPAdes, some assembly algorithms are able to capture contiguity data that others cannot. Also, certain classes of assemblers
may not be ideal for all sequencing technologies and resultant data profiles. Thus it is worthwhile to explore outside the scope of single assembly pipelines, as a meta-algorithmic approach through this type of merging can be advantageous.
Chapter 5

Discussion

5.1 Future

The Assembly RAST server is deployed and running. Useful metadata is being collected which will aid greatly moving forward. Most importantly, a foundational framework and functional server puts us in position to pursue and answer further questions. The following list describes intentions for future research:

Pipeline Result/Error Analysis and Classification: The AssemblyRAST framework will allow for simple and rapid invocation of multiple workflows and parameter sweeps, with the convenience of automated assembly assessment. We aim to produce a comprehensive analysis of algorithms, data sources, pipelines, and parameter optimizations, to elucidate strengths and faults, and understand key relationships between each respective level. This will allow for intelligent approaches to assembly strategy.

Hybrid Assembly: These assembly methods, whether using multiple algorithms or multiple sequencing technologies, is a relatively unstudied area although the few surveys of this idea seem to show promise. Furthermore, few tools have been explicitly developed for this strategy, so pursuing the topic further is worth our time.
Accurate Hands-Free Automatic Assembly: Allow users to submit genomes or metagenomes without explicit instruction, and through introspection of datasets and/or with explicit information from the user, formulate an optimal pipeline of processing stages for assembly.

Integration with Other Services: Once genomes are properly assembled, users are able to further garner insights using additional forms of analysis. Thus, an automated transfer to applicable services would be highly beneficial for scientific throughput. Our near-term goals are to integrate with the RAST and MG-RAST servers for annotation of contigs.

Eukaryote Assembly: Microbial and metagenomic assembly are currently the focus of the system. As we improve assembly accuracy and throughput, support for eukaryotic genomes will develop. This will be helpful to such efforts as the Genome 10k project.

Metagenomic Communities: Metagenomic assembly remains a prominent challenge due to the complexities inherent in the data provided. To produce better methods of assembly, it will be necessary to better understand properties of microbial communities, studying abundance profiles, clustering algorithms, phylogenetic analyses, and a myriad other facets.

Performance Profiling: Currently, computational biology tools and services are built and executed on standard computer service infrastructures. Building out a comprehensive benchmarking framework to analyze performance and characteristics of computational biology tools will provide insight into the development of novel computational architectures better suited for computational biology.

Data reuse mechanisms: It is a clear assumption that disk I/O is a major bottleneck to the computational biology problem. Complex pipelines are generally composed
of separate UNIX processes and therefore pass on intermediate states through disk I/O. Further investigation into efficient inter-process data passing of large sizes is necessary.

**Robust services**: Improvements on the service-oriented framework is necessary to sustain reliability as a publicly available resource. Assembly RAST is currently built upon the “industry-standard” tools; the overall heterogeneity of workflow, tool invocations, and data types incite further investigation into alternate approaches. Globus Online is a near-term goal for implementation.

**Alternate Compute Architectures**: Novel architectures may be capable of accelerating certain computations. For example, Convey’s HC-1 machine uses an FPGA co-processor to accelerate specific task; we are currently testing a BWA read alignment implementation on this machine. The flexibility of the Assembly RAST compute framework allows for a heterogeneous mixture of compute systems, and thus alternate systems can be used.
Bibliography


